

All About FORTH

An Annotated Glossary

Glen B. Haydon

Third Edition

REVISED AND UPDATED

Includes

COMMON USAGE

STANDARDS DOCUMENTATION

FOUR IMPLEMENTATIONS

All About FORTH

An Annotated Glossary

Completely Revised and Updated
with
Common Usage

Includes
MVP-FORTH for the IBM computer family
fig-Forth for the 8088/86 CP/M86
Selected F83 Source
Selected F-PC Source

Appendix
fig-Forth Installation Manual (Less 6502 source)
Forth-79 Standard
Forth-83 Standard

by
Dr. Glen B. Haydon
Box 429, Route 2
La Honda, CA 94020

Third Edition

June 1990

MVP-FORTH SERIES

Dr. Glen B. Haydon, General Editor

Volume

1. All About Forth, Glen B. Haydon, 1990
2. MVP-FORTH Source Listings, Glen B. Haydon and Robert E. Kuntz, 1983
3. Integer and Floating Point Math, Philip Koopman, Jr., 1983 (Revised)
4. Expert System, Jack Park, 1984
5. File Management System, Pierre Moreton, 1984
6. Expert Tutorial, Mitch and Linda Derick, 1984
7. FORTH Guide, Glen B. Haydon, 1985
8. IBM Professional Applications Development System, Thomas E. Wempe, 1985
9. Word Processor and Calculator Development System, Programmer's Guide, Thomas E. Wempe, 1985
10. Word Processor and Calculator Development System, File & Print Source, Thomas E. Wempe, 1985

Copyright © 1990 by Glen B. Haydon

The contents of this publication are released without restrictions.

Acknowledgement on subsequent use would be appreciated.

Table of Contents

Dedication	v
Acknowledgements	vi
Preface To The Third Edition	vii
Introduction	ix
The Inner Interpreter	xv
Notations	xxiii
Glossary	1
fig-Forth Installation Manual	A 1
Forth-79 Standard	B 1
Forth-83 Standard	C 1
Forth-79 Handy Reference	D 1
Index	I 1

Dedication

I dedicate this Third Edition to the thousands of MVP-FORTH users world wide. All versions of the program have been stable since 1983 and will remain so. This volume merely updates and expands our continuing support.

Acknowledgements

Charles Moore wrote the initial versions of Forth. Much credit goes to him for bringing the ideas of the language to life. William Ragsdale has led the Forth Interest Group in making the language available on some 12 CPU's. They have placed all the implementations in the public domain. There are just too many in the Forth Interest Group who have contributed to acknowledge them individually, so I acknowledge them all collectively.

I must acknowledge the valuable help of several people in developing MVP-FORTH and its original documentation. Jerry Boutelle adapted his *Nautilus Systems Cross-Compiler* to generate a 79-Standard system and gave me many hours of tutorial in understanding the language. He collaborated in the implementation of MVP-FORTH and proofread this manuscript. Robert L. Smith has contributed to my interpretation of the 79-Standard and carefully reviewed my implementation of the Required Word Set. Klaxon Suralis read the manuscript letter by letter and made many editorial and substantive suggestions. Roy Martens has given me the added push necessary to bring all this work together.

I thank those who have taken the time to point out specific difficulties with MVP-FORTH and previous editions of this manual. As a result, I have been able to write many clarifications. I also thank those who have pointed out the inevitable typographical errors. I hope that the new edition does not add more than it corrects.

Those who have helped in developing this Third Edition deserve special acknowledgement and thanks: Jerry Lilly, John Dellis and Roger Wallace helped define the scope of the work. Mike Perry helped interpret the *Forth-83 Standard* and parts of *F83*. Tom Zimmer helped interpret parts of *F-PC*. Marlin Ouverson helped in the review of *Forth Dimensions*. Jonathan Ross read the entire document and made many editorial suggestions.

I have not always adopted the generous suggestions that they offered, but I appreciate the thought and effort that has gone into them. I remain fully responsible for the result.

Preface To The Third Edition

The Third Edition has become a concordance of public domain functional definitions of ideograms (Forth words) in common usage: The *fig-Forth Model*, *The Forth-79 Standard*, *MVP-FORTH*, *The Forth-83 Standard*, *F83* and *F-PC*. In addition the 8086/88 implementations for *fig-Forth*, *MVP-FORTH*, *F83* and *F-PC* are also included. These entries are taken verbatim. Several sources were available in ASCII format which I could merge with the previous edition of *All About Forth*. The remainder were added by hand. I have verified the result against each source. In spite of all efforts I fear that there will be a few errors. I have corrected minor typos. To this concordance, I have added examples and comments for each included entry.

Many dialects of Forth have an associated glossary. The commercial implementations pride themselves in their glossaries. But none of them are much help in understanding the code in a different dialect. Furthermore, the written functional definitions do not always describe the associated implementation. Too often the commercial implementations are proprietary and you have no way of knowing what is happening. Sometimes the same functional definitions will have different implementations behaving differently and other times the implementations may be the same but have different functional definitions. This is the advantage of a completely open language. But it does lead to problems in understanding Forth.

The problem was the same when I first developed MVP-FORTH. At first, *fig-Forth* was all that was available. Then after rumors that it was going to be withdrawn, I modified the source code to implement the 79-Standard. About 40 ideograms in *fig-Forth* needed to be changed and many were dropped, including most of the very necessary primitives. The result was a "pure" 79-Standard implementation and it imposed a most frustrating limitation, especially after one had become used to *fig-Forth* where everything was available. Then came the book, *Starting Forth*, which was the first instructional text on the use of the language. Though it made reference to the 79-Standard, the function of many of its ideograms (Forth words) is based on poly-FORTH, whose implementation is not in the public domain. Even then, the differences were a blow to writing portable user programs. Over the years the problem has gotten worse in spite of the 83-Standard. This Annotated Glossary of Common Usage is a result of my efforts to keep up with the changes.

Forth cannot be mastered overnight. Learning from a reference manual

can be difficult. The tutorial *Starting Forth* First Edition, by Leo Brodi, Prentice Hall, 1981, is still an excellent adjunct to the learning process. A number of other books and tutorial programs are available. The volumes in the *MVP-FORTH Series* are designed to help learn MVP-FORTH. However, nothing can replace active use of Forth on a system.

The MVP-FORTH implementations included in this Third Edition have been changed to their most common version: that for the IBM PC and its clones. The only difference among the current MVP-FORTH versions is in the interface with the system. This is the same system as that selected for F83 and the only system for F-PC. The dialects are now easy to compare.

The section on the Interpreter has been edited. I appreciate the help and suggestions from Charles H. Moore and Kim Harris in developing this discussion.

For those interested in the technical details of the procedure I used: The CP/M ASCII text files for the second edition of *All About Forth* were ported to an IBP PC clone; the files were filtered with a Forth program to be compatible with XYwrite and Ventura. F83 source files were converted to text files. In several windows of XYwrite, these sources were loaded and block moves were used to merge selected portions into the master text file. The data not available on disk, was entered by hand. The first ten volumes of *Forth Dimensions* were read and selected functions not already present were added along with data from the other sources available. No attempt has been made to include all the functions of either F83 or F-PC. The work is already large. Each concordant part of each entry has been checked character by character with the original. Ventura was used to format the glossary in a single chapter! The implementations have been reformatted from the original. This work has been an education for me. I hope it will be useful to others.

If you are confused about some of the entries you might refer to the section on Entry Organization. Good luck!

GBH
June 1990

Introduction

Forth is a relatively new computer language. Dialects of Forth are increasing in number. This glossary has been assembled to include the more common usages in the hope that it will serve as a reference and prevent future dialects from deviating too much from common usage.

This Third Edition is directed to MVP-FORTH users to whom this work is dedicated. Though not used in MVP-FORTH, new definitions in the *Forth-83 Standard* Required Word Set and a generous selection of functions and implementations now in common usage are included. The reader can thus understand new Forth programs as he comes across them and adapt them to his system.

Ideograms

I use the word "ideogram" to characterize the names of the functions used in Forth and to emphasize their nature. These ideograms are frequently called words in Forth, but the ideas associated with each ideogram are somewhat different from words in the usual context within the Indo-European cultures. I refer you to my article, "Forth and the Nature of Ideographic Thought" (1981 Rochester Forth Standards Conference) for a discussion of this usage of ideograms. There is no relation between many of the symbols and written words in English. The use of ideograms provides a mind-set in using Forth.

Levels of Forth

Level 0 includes the 63 functions which Charles H. Moore has often listed as the basis of Forth. They lack any form of input or output to storage devices.

Level 1, fig-Forth, adds a rudimentary set of storage input and output functions. These functions extend Forth to a form of archival memory. It can be used to compile run-time programs. However, the *fig-Forth Installation Manual* did not include a way to write to the archival memory; only to access it. Later printings of the manual included a source listing for an editor, but no way to enter it on to disk without compiling it first from the terminal. (See Appendix A).

Level 2, MVP-FORTH, adds several functions making Forth its own operating system and providing a small efficient working environment. A line editor, assembler, utility and supplemental functions are added to support the first edition of *Starting Forth* as a tutorial for beginners. Including all the primitives and extensions, about 240 functions are

included. Most applications can be written using about 100 of the most common functions. The functional definitions are those defined in the *Forth-79 Standard* (See Appendix B)) where they superseded fig-Forth. MVP-FORTH is available for many processors. It remains popular and is supported with the ten volume *MVP-FORTH Series*.

Level 3, F83, modifies and reinterprets the 132 functions in the *Forth-83 Standard Required Word Set* (See Appendix C). It adds many more functions to make the total approximately 1000. There are slight differences in interpretation from the 83-Standard - hardly noticeable. Implementations have been written for many processors. The much expanded system now uses Forth screens in system files. Many new features provide excellent examples. The documentation is limited to shadow screens conveniently linked to the source screens. For the Intel based systems, programs are limited to a single segment of 64K bytes.

Level 4, F-PC, is an outgrowth of F83 with many enhancements. It has gone through several years of evolution and is currently up to Version 3.5. The implementation is limited to Intel based systems running under PC-DOS or MS-DOS. It remains a 16-bit system, but makes use of multiple segments extending the limits beyond the 64K segment boundary. Forth screens are replaced with ASCII text files. An ingenious hyper-text like technique provides excellent on-line assistance. Several publications are available. It provides a fully integrated Forth programming environment. The program is large, requiring at least 2 megs of hard disk space to run as designed. The number of functions available has increased to about 2500, about ten times the size of MVP-FORTH. It is fast and provides an ideal platform for Forth programmers working on an Intel based system.

Level 5, is in the future. Several models of Forth implementations with 32-bit linear address space, 32-bit stacks, and 32-bit operators are available. In these implementations, most of the Level 0 functions are the same but now operated on 32 bits instead of 16 bits. The day will come when 32-bit linear addresses will be available even in the segmented memory of the Intel processors. It remains to be seen how these implementations will be accepted by common usage.

The Future

The ANS Technical Committee is making a major revision of Forth functions. It will be at least another year, if not two, before their work will be officially adopted. It will take a couple more years before we will see how much of their work comes into common usage.

A most interesting new world of Forth is just beginning: the hardware implementations of stack machines. Phil Koopman's book, *Stack Com-*

puters - *The New Wave*, Ellis Horwood Limited & Halsted Press division of John Wiley & Sons, 1989, is an excellent reference. Real-time applications are just beginning to use these processors.

Other Publications

Other implementations of Forth have also published glossaries, but often without the details of the actual implementation. Among these, an early one is from the National Radio Astronomy Observatory, Tucson, Arizona, *Computer Division Internal Report No. 17, Basic Principles of Forth Language as Applied to a PDP-11 Computer*, by E. D. Rather, C. H. Moore and Jan M. Hollis, March 1974. The Kitt Peak National Observatory, Tucson, Arizona 85726, published *A Forth Primer* by W. Richard Stevens, February 1979 (Update 2). In addition, several Universities have implemented Forth. Among them are the University of Utrecht, the Netherlands, and the University of Rochester, Rochester N.Y. The glossaries associated with these versions of Forth have been made available without copyright limitations and are in the public domain. After the latest version of MVP-FORTH, the Forth Standards Team published the *Forth-83 Standard* August 1983 (See Appendix C). It too is available from the Forth Interest Group. More recently, the ANS ASC X3/X3J14 Technical Committee has published quarterly an update of their work. Their latest meeting was in May 1990. Their publications are labeled as Preliminary Documents and Subject To Change. There are currently 12 such preliminary revisions. Their documentation is distributed by the Committee at 111 N. Sepulveda Blvd, Suite 300, Manhattan Beach, CA 90266. Because this work is preliminary and has not been released for common usage, it has not been included in this work.

MVP-FORTH

The definition of each ideogram gives precedence to that given by the Forth Standards Team, as given in their publication *Forth-79 Standard* (October 1980). It is distributed by the Forth Interest Group whose current address is: P.O.Box 8231, San Jose, CA 95155. These definitions have been placed in the public domain. A second source is the *fig-Forth Installation Manual Glossary Model* (May 1979). It is also distributed by the Forth Interest Group. This publication includes in addition to the glossary, an implementation of that glossary for the 6502 processor. It is placed in the public domain with the statement that further distribution must include a notice. The *fig-Forth for 8080 Assembly Source Listing*, also from the Forth Interest Group, has served as the model for the internal structure of MVP-FORTH as it is now implemented. (The original version of MVP-FORTH was written for the Intel 8080 processor.) The contributions of the Forth Interest Group have been a major stimulus to the spread of the language.

All of the *Forth-79 Standard* Required Word Set is included as faithfully as possible. I have added some ideograms in conjunction with the underlying operating system and several additional utilities that I find helpful. The primitive ideograms used in defining the *Forth-79 Standard* are also included.

I found two major stumbling blocks in earlier versions of Forth. One is the use of parenthesis for comments as well as primitives, and the other with double number operators. My modifications are noted as follows:

1. The use of parentheses around ideograms to show that they are primitives conflicts with the use of parentheses around comments. One cannot refer to such a primitive within a comment without terminating the comment. I have adopted the convention of placing such primitives within angle brackets. For the sake of completeness, both forms are included in the glossary, but only the angle brackets are used in the MVP-FORTH implementation.

2. I feel that the convention of combining the ideogram, 2, with other ideograms is frequently confusing. This ideogram conveys the idea of quantity, such as in 2+ , 2* and so on. Originally, Charles H. Moore did mean two constants with 2CONSTANTS. He was using ratios in his calculations. On the other hand, the ideogram D is combined with other ideograms to show that they apply to double precision numbers such as DMAX, DMIN, DNEGATE, etc. I find the ideograms in the *Forth-79 Standard* Extension Word Sets headed 11.1 Double Number Word Set, using the ideogram, 2, to be confusing. True, it makes little difference in some cases, and the use of synonyms is appropriate; but a double number word set should have the connotation of double numbers associated with them. Thus, I have made the basic definitions of all words in that set start with D, rather than the 2 that some of them now have. There are some systems in which the ideograms beginning with 2 and D will have different implementations. Beware! I have found one implementation of 2DUP which places two copies of the top element on the stack. For those already in the habit of using the original definitions, and for possible applications already written, I have implemented the ideograms as synonyms.

Editor

Editor ideograms are not included; however, two rudimentary editing ideograms are included: PP and CLEAR. They make it possible to enter and load your choice of editors from the published listings. Several good line editors and screen editors are available, some of which are in the public domain. For convenience, the source code for the line editor from *Forth Dimensions*, Vol. III, No. 3 is included. It can be loaded on top of MVP-FORTH. Remember, when using an editor, it is necessary to call the

vocabulary, usually named EDITOR. *Starting Forth* does not make this clear in chapter 3.

Assembler

Source listings for a Forth Assembler based on that published by Ray Duncan in *Dr. Dobbs Journal*, is included with MVP-FORTH on the screens disk. It can be loaded on top of MVP-FORTH.

Utilities

The source listings for a set of utilities are also included. They can be loaded on top of MVP-FORTH. They make it possible to save a run-time program among many other things.

Supplementals

The functional definitions given in *Starting Forth* summarize definitions from many sources, all in the public domain. Unfortunately, there are some conflicts in the function of some of these ideograms with the 79-Standard. In MVP-FORTH, these have been decided in favor of the 79-Standard. Those functions not already included, can be loaded from the Supplementary source screens.

Summary

Among the distribution files, MVPFORTH.COM is the assembled source code in object form. FORTH+++.COM (I ran out of pluses) has the Editor, Assembler, Utilities and Supplemental source added. The source for these is on a second disk. In some distributions packages, the Forth source screens are compressed and stored as a file with the rest of the programs. Instructions for making a Forth screens disk is included when that is necessary.

The Inner Interpreter

The key to Forth's power lies in the efficiency of its inner interpreter. This section will help you appreciate the power of the language.

Forth provides an interface for application software to a variety of processors. When a common dialect of Forth is implemented on different processors, all high level Forth programs are portable among them. The implementation of Forth's inner interpreter is processor dependent. A generic description of the necessary functions can be written in Forth.

This discussion is based on indirect threaded code. Other implementation may use direct threaded or token threaded Forth.

The inner interpreter determines the order of execution of Forth functions. The most common functions are a series of other Forth functions that are compiled in colon definitions. Such compiled definitions contain an array of addresses performing the defined function when executed in sequence. Colon definitions can nest other definitions without limit. Of course the size of memory does impose a limit on the space in the dictionary for new definitions. The inner interpreter finds its way through all levels of nesting using pointers.

The five necessary pointers are described below. For a given processor, these pointers may be assigned registers or memory addresses. We will assign these pointers to Forth variables and describe the implementation of the necessary functions in high level Forth.

All processors have a program counter. In our generic Forth system we make the program counter a variable (PC). Our implementation of Forth uses two stacks. We use a separate pointer for each of the stacks: the data stack pointer, (SP), and a return stack pointer, (RP). The data stack is used for temporary storage of data and for operations on data. The return stack is used to keep track of the sequence of nested functions for the inner interpreter. We must keep up with two other Forth pointers: the interpreter pointer, (IP), and the current word pointer, (W). IP points to the next word to be interpreted. W points to the word currently being executed. The key to the function of Forth lies in the interaction among these pointers.

Variable Pointers

We define these pointers as variables in our generic system.

VARIABLE	PC	\ Program counter
VARIABLE	SP	\ Data Stack Pointer
VARIABLE	RP	\ Return Stack Pointer
VARIABLE	IP	\ Interpreter Pointer
VARIABLE	W	\ Current Word pointer

The inner interpreter is contained in a group of instructions that we will describe in high level Forth.

Next

We will begin with the interpretation of definitions. The dictionary header for each definition includes a code field. Execution of a function is initiated by an indirect jump to a group of machine instructions being pointed to by the code field. These machine instructions, and in fact all machine instructions, end with a jump to the next group of machine instructions to be executed. The interpreter pointer keeps track of the sequence of pointers and parses the series of addresses contained in the definition's parameter field. These addresses point to the successive operations of a definition.

The last operation concludes with a jump to the beginning of NEXT. It gets the next address value from the IP and moves it to the PC. This function is factored into two parts the bottom one of which is:

```
: NEXT1
  W @ @    PC !  ;
```

The address pointed to by W is moved to the PC. The new address in the PC becomes the address of the next opcode for the system.

Note: The address in IP must be incremented at some point.

NEXT starts the operation and ends with NEXT1:

```
: NEXT
  IP @ @    W !    2 IP +!    NEXT1  ;
```

NEXT fetches the address currently being pointed to by the variable IP and stores it in W. The address in IP is incremented 2 bytes to set it to the next address. Finally, the function of NEXT1 places the address pointed to by W into the system's PC.

Colon Definitions

The code segment pointed to by the code field instructs the program in it's handling of the information in the parameter field. The code field for

a colon definition points to a machine code segment we will name DOCOL. This name is not a usual Forth ideogram and is not directly accessible in most Forth implementations. DOCOL's function is to interpret the series of code field addresses that follow.

DOCOL may be factored. The factor, DOCOL1, illustrates the use of a variable as a stack pointer. Remember, a stack pointer points to the current value on the top of a stack. When a value is added to the top of a stack (pushed), the stack pointer must be moved to the new address before the value is stored there. Also, after a value is removed from a stack (popped), the stack pointer must be moved to the previous value on the stack.

DOCOL1 saves the contents of IP on the return stack.

```
: DOCOL1    -2 RP +!    IP @    RP @ !    ;
```

The content of RP is first decremented 2 bytes, one stack cell, in preparation for pushing a new value. Then the current address in IP is fetched and placed in the new cell on the return stack.

Then the complete function at DOCOL can be defined:

```
: DOCOL    DOCOL1    W @ 2+    IP !    NEXT    ;
```

After storing the current value in IP on the return stack, the value in W is fetched and incremented by 2 to point to the next word to be interpreted. It is then stored in IP and NEXT is executed.

A colon definition is terminated by executing EXIT in most Forth's vocabularies. Rather than continuing through the address cells in the parameter field of a colon definition, the top of the return stack is popped to IP before NEXT is executed.

```
: EXIT    RP @ @    IP !    2 RP +!    NEXT    ;
```

The value currently pointed to by RP is fetched and stored in IP. RP must now be changed to point to the next value on the return stack. Then NEXT is executed. In fig-Forth, the ideogram, ;S, was used.

Data Stack

We will need to push and pop values to our generic data stack. The data stack in our descriptive Forth, might be thought of as the accumulator for our system. We define a PUSH and POP. These are not a part of Forth's standard vocabulary. By factoring these functions, the subsequent discussion is much simplified.

```
: PUSH    -2 SP +!    SP @ !    ;
```

Move contents of SP to point to a new value, fetch the address of the new location and store the value.

```
: POP      SP @ @      -2 SP +1 ;
```

Fetch the value on the top of the data stack and then move contents of SP to the previous value.

Execute

It is possible to interrupt the normal sequence of the inner interpreter and execute the address on the top of the data stack by EXECUTE:

```
: EXECUTE   POP  W !   NEXT1  ;
```

The value currently on the top of the stack is moved to W. The opcode pointed to by this address is then executed by NEXT1. If the location pointed to by the address on the top of the stack is not machine code, the results are completely unpredictable.

Variables

Execution of a variable places its parameter field address on the data stack. The function pointed to by the code field address of a variable, is often named DOVAR. The name is not an ideogram in most Forth implementations.

```
: DOVAR     W @      2+   PUSH   NEXT  ;
```

The address in W is incremented 2 bytes and is pushed to the data stack. NEXT is then executed. This function places the address of an integer variable on the top of the stack. It is the address of the beginning of the parameter field.

Constants

Execution of a constant places its value on the data stack. The function pointed to by the code field address of a constant, is often named DOCON. The name is not an ideogram in most Forth implementations.

```
: DOCON     W @ 2+ @   PUSH   NEXT  ;
```

The address in W is incremented 2 bytes, its content is fetched and pushed on the top of the data stack. NEXT is then executed. This function places the contents of the parameter field on the top of the data stack. It is the value of the constant.

User Variables

Execution of a user variable places its address on the data stack. User variables are accessed as an offset relative to the beginning of the user table. This is accomplished by a code routine which is often named DOUSE.

```
: DOUSE  
  W @ 2+ @      UP  +      ( Add base of user table.)
```

PUSH NEXT ;

The contents of the cell following that pointed to by **w** is fetched. **UP** is a constant (it may be a variable in some systems) that is the address of the base of the user table. The values are added and pushed on the data stack. It is the actual user variable address. **NEXT** is then executed.

DOES>

One of the most unusual capabilities of Forth is the ability to write new defining ideograms. A defining ideogram is used to compile new entries into the dictionary with a new and unique function. That function is usually defined in high level Forth following the ideogram **DOES>**. A short code operation is called to initiate the high level definitions. The defining ideogram's definition is a standard colon definition. It is then used to define new ideograms.

The defining ideogram uses a call, or a jump to a subroutine that in most processors places the address for the return function on the top of the hardware stack. Remember, in our system the hardware stack and the data stack are the same. The called subroutine will place a newly defined ideogram's parameter address on the data stack and then execute the code field addresses for the functions that follow **DOES>** in the defining ideogram. The subroutine that is called is often named **DODOES**, but it is not included in the Forth vocabulary.

This Forth operation is one of the most difficult to understand. It might help to construct a diagram showing the memory image of each of the three words: **DOES>**, the new defining word, **<name>**, and a new word being defined with it, **<namex>**. Then trace the operation of each word.

```
: DODOES  
  DOCOL1                    ( Push IP on the return stack )  
  POP IP !                ( Store the call's return address )  
  W @ 2+ PUSH            ( Push the address of the next cell )  
  NEXT ;
```

DOCOL1 saves the current interpreter pointer on the return stack. Remember, a call routine including the call to **DODOES** will leave the address of the cell following the call on the systems hardware stack and we have made that the data stack. We pop the return address and store it in **IP**. We then fetch the address in **W**, increment it by 2 and push it onto the data stack. Finally, we execute **NEXT**.

Besides these several names pointed to by the code field of entries in the dictionary, several other machine instructions are useful. They include a method of including a value in a colon definition that is to be

pushed on the data stack, and two branch routines.

Literals

In colon definitions a series of code field addresses is usually compiled. Still, it is occasionally desirable to compile an integer value that is to be placed on the stack at run time. This requires that the preceding code field address places the contents of the next address on the data stack and then skip to the next address in the parameter field. This function is done with the Forth ideogram `LIT`.

```
: LIT  IP @ @  PUSH      2 IP +!  NEXT  ;
```

The value pointed to by the address in `IP` is fetched and pushed onto the data stack. `IP` is incremented 2 bytes to skip to the next code field address before going to `NEXT`.

Branches

Structures such as `IF ... ELSE ... THEN` and `BEGIN ... UNTIL` require conditional and unconditional branches. They require a change in the interpretive pointer for the next execution. These can be done with the designation of an absolute jump or a relative jump from the location of the code. The relative jump is more common.

The unconditional branch is easiest. It is often named `BRAN` and may be defined:

```
: BRAN  IP @ @  IP @ +  IP !  NEXT  ;
```

Fetch the contents at the address pointed to by `W`. It is the offset. Then fetch the address itself and add them. This will be the address to which to jump. Store it in `IP` and execute `NEXT`.

The conditional branch, `0BRANCH` is often named `ZBRAN` and may be defined.

```
: ZBRAN
  0=  IF  BRAN  ELSE  2  IP  +!  THEN  NEXT  ;
```

If the value on the stack is 0, set the flag left on the stack or else reset it. Then if the flag is set, i.e., value was a zero, execute `BRAN`. If the flag is reset, then skip the cell containing the offset value and start execution with the next cell.

Code Definitions

Finally, we need an address to place in the code field for a code definition using the assembler. This is simple. Remember that the code field address is a pointer to machine code. The machine code begins at the next address. This is done with `CODE` in Forth.

```
: CODE
```

CREATE HERE DUP 2- ! ASSEMBLER [;

After CREATE in the definition of CODE, the sequence, HERE DUP 2- !, places the next address in the code field address of the new ideogram. This address is the beginning of the parameter field that contains machine code. It must end with a jump to NEXT.

Summary

This discussion uses English and Forth to describe the heart of the language, the inner interpreter. Forth provides a compact and precise notation for describing the necessary actions. Ultimately, it will have to be implemented in machine code for a real processor to run a Forth system. You will have gone a long way to understanding Forth when you understand this section.

Notations

All of the required words in the 79-Standard are included with the functions as defined by the Standards Team. The required words can be cross-compiled alone to make a "pure" implementation of the 79-Standard that includes none of the primitives or extensions. To my reading, the standard implies that all added vocabularies must be programmed from the Required Word Set. This includes those functions in the Double Number Word Set, the Reference Word Set, and all application programs. A "pure" 79-Standard program should comply with these requirements.

I find the standard's requirements frustrating. MVP-FORTH is not strictly standard, though all the individual functional definitions comply. MVP-FORTH has been cross-compiled to include all the primitives and selected 79-Standard extensions, and some additional utilities. Source for the implementation of all but a few obsolete ideograms are included in the glossary and the full implementation source is included in the MVP-FORTH product. A 79-Standard version can be simply cross-compiled with all non-standard ideograms left headerless, but it is useless.

The glossary of Forth ideograms is listed in the numerical sequence of the ASCII codes used in the ideograms. Where the definition of the ideogram is used in the 79-Standard, that definition is given. Then, as necessary, definitions are taken from the Extension Word Set and Reference Word Set, fig-Forth; and finally a few of my writing. Though most of these have the same function, not all of them do.

Entry Organization

The entries in the glossary are listed in ASCII order. Each entry includes the following:

1. The Entry
2. The Stack Notation before and after execution. There has been no consistency in the stack notations among the various sources. I have continued to use the notations from MVP-FORTH. I have left in the original, the text from other sources may refer to the stack with different notations. I leave the interpretation to the reader. For example the following notations may all refer to the same type of stack entry: *n*, *n1*, *un*, *u*, *adr*, *addr*, *first*, and *last*.
3. A designation concerning its use:

MVP-FORTH These entries include the entire Required Word Set from the *Forth-79 Standard* as well as most of the primitives necessary to implement them. Together, they are included in the source for the

cross-compiler and in the assembly source listings. This combination constitutes the MVP-FORTH nucleus. Everything except the Required Word Set may be compiled as headerless ideograms to produce a "pure" 79-Standard.

MVP-FORTH DISK I-O These entries for the system disk access routines are also included in the MVP-FORTH nucleus, but they will not be the same for all implementations.

MVP-FORTH UTILITY These entries include a selection of utilities. Their source is included as a set of screens that may be loaded with the MVP-FORTH nucleus to provide a convenient MVP-FORTH Development System.

MVP-FORTH SUPPLEMENTAL These entries include those additional definitions necessary to implement some older ideograms and the vocabulary used in *Starting Forth*. Note that some *Starting Forth* functions conflict with 79-Standard functions. These will not execute according to *Starting Forth* usage. For example, many students have lost much time trying to make the examples of .S run with fig-Forth or 79-Standard systems. It is just not possible.

NOT USED These entries are included for reference only. They include all the new ideograms in the 79-Standard required word set and a selection of other ideograms in common usage. Should one wish to use them in MVP-FORTH, he must add them. Usually, the necessary implementation is given.

4. The Form of the use of the ideogram where appropriate.

5. The Pronunciation of the ideogram where appropriate.

6. The Implementation may be given in high level Forth, or in assembly language for the Cross-Compiler. Note that unless otherwise shown all numerical values are given in hex and that some labels refer to words only available in the Cross-Compiler. The actual implementation in MVP-FORTH may differ, but the functional definitions are adhered to. Where different, the implementation from Thomas Newman's implementation of fig-Forth for the CP/M-86, Mike Perry and Henry Laxon's F83 Version 2.4.0 Modified 25 January 1989, and Tom Zimmer and Associates' F-PC, Version 3.5 are used.

A special word needs to be said about the assembly code used and its format. Some of the implementations use assembler macros. These are most obvious at the end of code segments. Thus NEXT, NEXT JMP, JMP NEXT may include the necessary END-CODE functions. The form included in each case is that of the particular implementation. However, where this is the only difference between implementations, they have been

declared the same. Most Forth assemblers use reverse Polish notation. F-PC allows either reverse Polish, or the more familiar Intel assembler notation. Its source code is listed in the more familiar format.

7. The Example may be an important or representative definition from the source code, a reasonable use, or in a few cases a contrived use. With each example is a short discussion of the example.

8. The Comments try to put that ideogram in a perspective.

9. Occasional notes have been added at points of particular concern or where caution must be used.

Summary

Most of these comments are obvious. Good Luck!

The Glossary


```
: !CSP  SP@  CSP  !  ;
```

Same as MVP.

Same as MVP.

Same as F83.

!CSP

Comment: This ideogram conspires with ?CSP to trap malformed colon definitions at compile time. It is used in fig-Forth; error checking is not standardized and may vary widely among implementations. Although the fig-Forth approach is used here, this ideogram is next to useless; MVP-FORTH omits it, spelling out SP @ CSP ! as required.

MVP-FORTH

Pronounced: store long

CODE IL

```

BX POP    DS POP    AX POP    [BX], AX MOV
BX, CS MOV DS, BX MOV NEXT JMP  END-CODE

```

Not used.

Not used.

CODE !L

```

POP BX    POP DS    POP AX
MOV 0 [BX], AX    MOV BX, CS    MOV DS, BX
NEXT      END-CODE

```

Example:

```

HEX
4182 2000 0 !L
DECIMAL

```

Store the 16-bit value, 4182, in segment 2000, address 0.

Comment: : The ideogram is used with the 8086/8088, 16-bit CPU. This CPU utilizes segments for the extended addressing. In MVP-FORTH for the 8086/8088, all segments begin at the same location.

" | <string> --- NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined (F83) Compile the string, return pointer later. (F-PC) Compile the string into CODE space. Return the address and length of the string at run time.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```

: "
  COMPILE (") , " ; IMMEDIATE

```

F-PC:

```

: "
  COMPILE (") HERE X, , " ; IMMEDIATE

```

Example:

```

: GREETING " Hello" ;

```

GREETING is defined. Upon execution, GREETING will now leave a count and an address on the stack ready for TYPE.

Comment: This ideogram will store a string. The count and pointer can be used to move or manipulate the string in many ways.

Note: Although a number of string functions have been suggested, there is no set in common usage.

#

ud1 --- ud2

MVP-FORTH

Generate from an unsigned double-number ud1, the next ASCII character which is placed in an output string. The result ud2 is the quotient after division by BASE and is maintained for further processing. Used between <# and #>. (*Fig*) Same as MVP. (*79S*) Same as MVP. (*83S*) The remainder of +d1 divided by the value of BASE is converted to an ASCII character and appended to the output string toward lower memory addresses. +d2 is the quotient and is maintained for further processing. Typically used between <# and #>. (*F83*) Convert a single digit in the current base. (*F-PC*) Same as F83.

Pronounced: sharp

Implementations:

MVP: 8088/86:

```
HEX
: #
  BASE @ M/MOD ROT 9 OVER <
  IF 7 + THEN
  30 + HOLD ;
```

Fig: 8088/86:

```
HEX
: #
  BASE @ M/MOD ROT 9 OVER <
  IF 7 + ENDIF 30 + HOLD ;
```

F83: 8088/86:

```
: #
  BASE @ MU/MOD ROT 9 OVER <
  IF 7 AND + THEN ASCII 0 + HOLD ;
```

F-PC:

Same as F83.

Example:

```
43 0 <# # # # #> TYPE
```

The value 43 is extended to unsigned double precision by placing a zero on top. Using the current value of BASE, the numeric conversion ideograms then extract the three low-order digits, 043, passing them as an ASCII string to TYPE.

Comment: The fig-Forth definition of #, used herein, complies with Forth-79. The number is unsigned. If the value is to be signed, additional treatment is required.

#LINE

Implementations:

MVP: 8088/86:

NBUF CONSTANT #BUFF

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: SAVE-BUFFERS     #BUFF   1+   0
DO 0 BUFFER DROP LOOP ;
```

There is one pass through the loop for each buffer.

Comment: This constant simplifies the disk I-O code. Not all implementations have it. In MVP-FORTH, its value may be altered and then CHANGE will dynamically reconfigure memory to include the new number of buffers.

#LINE

--- a1

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined (F83) Number of CR's sent since last page. (F-PC) A variable that holds the line number of the current line on which text is being typed. Incremented by CR.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

VARIABLE #LINE

F-PC:

Same as F83.

Example:

```
#LINE @
```

The example returns the number of lines already typed to the current

page.

Comment: #LINE is used in print functions to monitor the number of lines already typed on the current page. Not all dialects make use of this variable.

#OUT --- a1 NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Number of characters sent since last CR. (F-PC) A variable that holds the column number of the most recent type or emit to the display.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

VARIABLE #OUT

F-PC:

Same as F83.

Example:

#OUT @ .

Examine the present value in the variable, #OUT and print it.

Comment: In fig-Forth and MVP the value is referred to with the ideogram OUT and is a User Variable. In either case, OUT or #OUT, is useful for checking space remaining on a line of output.

#S ud --- 0 0 MVP-FORTH

Convert all digits of an unsigned 32-bit number ud, adding each to the pictured numeric output text, until remainder is zero. A single zero is added to the output string if the number was initially zero. Use only between <# and #>. (Fig) Generates ASCII text in the text output buffer, by the use of #, until a zero double number ud results. Used between <# and #>. (79S) Same as MVP. (83S) ud is converted appending each resultant character into the pictured numeric output string until the quotient (see: #) is zero. A single zero is added to the output string if the number was initially zero. Typically used between <# and #>. (F83) Convert a number until it is finished. (F-PC) Same as F83.

#THREADS

Pronounced: sharp-s

Implementations:

MVP: 8088/86:

: #S BEGIN # DDUP OR NOT UNTIL ;

Fig: 8088/86:

: #S BEGIN OVER OVER OR 0= UNTIL ;

F83: 8088/86:

: #S BEGIN # 2DUP OR 0= UNTIL ;

F-PC:

Same as F83.

Example:

43 0 <# #S #> TYPE

The value 43 is extended to unsigned double precision by placing a zero on top. Using the current value of BASE, the numeric conversion ideograms then extract the three low-order digits, 043, passing them as an ASCII string to TYPE.

Comment: This ideogram is one of a group which must be used together in formatting the output of numbers. # #> #S <# HOLD SIGN

#THREADS

--- n1

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) The number of separate linked lists per vocabulary. (F-PC) Return the number of threads used in the dictionary.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

#THREADS-T @ CONSTANT #THREADS

F-PC:

#TTHREADS CONSTANT #THREADS

Example:

#THREADS @ .

Will display the number of threads into which the dictionary is hashed.

'-FIND

of <name>. An error condition exists if <name> is not found in the currently active search order. (F83) Return the code field address of the next word. (F-PC) Return the code field address of the next word <name>.

Form: ' <name>

Pronounced: tick

Implementations:

MVP: 8088/86:

```
: '
  -FIND NOT
  ABORT" NOT FOUND"
  DROP [COMPILE] LITERAL ;
  IMMEDIATE
```

Fig: 8088/86:

```
: '
  -FIND 0= 0 ?ERROR DROP LITERAL ;
  IMMEDIATE
```

F83: 8088/86:

```
: '   DEFINED 0= ?MISSING   ;
```

F-PC:

Same as F83.

Example:

```
' MAX-DRV @ .
```

Find the parameter field address of the constant, MAX-DRV, fetch and print that constant's value.

Comment: This Forth-79 version searches only FORTH after CONTEXT. The fig-Forth implementation, however, searched CURRENT as well. This difference between the two implementations is subtle, yet pervasive.

CAUTION: *Starting Forth* users: do not EXECUTE the parameter field address returned by this implementation until you use CFA to convert it to the code field address. Otherwise, you will crash the system.

'-FIND

--- addr

MVP-FORTH

A user variable containing the address to be executed by -FIND. (Fig)

Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Pronounced: dash find

Implementations:

MVP: 8088/86:

HEX

16 USER '-FIND

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

'-FIND @ 2 + NFA ID.

Type the name of the function currently assigned to -FIND, assuming it is not headerless.

Comment: By vectoring the -FIND function, it is possible to dynamically change the implementation of that function. Thus it is not necessary to recompile the source to make such changes.

' ?TERMINAL

--- f

MVP-FORTH

A user variable containing the compilation address to be executed by ?TERMINAL. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

HEX

18 USER '?TERMINAL

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

' BLOCK

Example:

```
: ?TERMINAL    '?TERMINAL    @    EXECUTE ;
```

The example is taken from the source for ?TERMINAL. It vectors the run-time function to the code field address stored at ' ?TERMINAL.

Comment: Unfortunately, hardware differences do not permit all implementations of this ideogram to be exactly the same. Thus it is not fully portable among all implementations of MVP-FORTH. However, the differences are minor, and can be adjusted to easily.

' ABORT --- addr MVP-FORTH

A user variable containing the compilation address to be executed by ABORT. (*Fig*) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Pronounced: tic abort

Implementations:

MVP: 8088/86:

HEX

1A USER 'ABORT

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
'ABORT @ 2+ NFA ID.
```

Type the name of the function currently assigned to ABORT, assuming it is not headerless.

Comment: By vectoring the ABORT function, it is possible to dynamically change the implementation of that function. Thus it is not necessary to recompile the source to make such changes.

' BLOCK --- addr MVP-FORTH

A user variable containing the compilation address to be executed byBLOCK. (*Fig*) Not defined. (79S) Not defined. (83S) Not defined.

(F83) Not defined. (F-PC) Not defined.

Pronounced: tic block

Implementations:

MVP: 8088/86:

HEX

1C USER 'BLOCK

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

'BLOCK @ 2+ NFA ID.

Type the name of the function currently assigned to BLOCK, assuming it is not headerless.

Comment: By vectoring the BLOCK function, it is possible to dynamically change the implementation of that function. Thus it is not necessary to recompile the source to make such changes.

' CR

--- addr

MVP-FORTH

A user variable containing the compilation address to be executed by CR. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Pronounced: tic-c-r

Implementations:

MVP: 8088/86:

HEX

1E USER 'EMIT

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

'EXPECT

Example:

```
'CR @ 2+ NFA ID.
```

Type the name of the function currently assigned to CR, assuming it is not headerless.

Comment: By vectoring the CR function, it is possible to dynamically change the implementation of that function. Thus it is not necessary to recompile the source to make such changes.

'EMIT

--- addr

MVP-FORTH

A user variable containing the compilation address to be executed by EMIT. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Pronounced: tic emit

Implementations:

MVP: 8088/86:

HEX

20 USER 'ABORT

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
'EMIT @ 2+ NFA ID.
```

Type the name of the function currently assigned to EMIT, assuming it is not headerless.

Comment: By vectoring the EMIT function, it is possible to dynamically change the implementation of that function. Thus it is not necessary to recompile the source to make such changes. 'EMIT is particularly useful for redirecting terminal output to the printer.

'EXPECT

MVP-FORTH

A user variable containing the compilation address to be executed by EXPECT. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined.

(F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

HEX

22 USER 'EXPECT

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
EXPECT    'EXPECT @ EXECUTE ;
```

This example is taken from the source code for EXPECT. It vectors the run-time function to the code field address stored at 'EXPECT.

Comment: With the development of some applications, it is desirable to change EXPECT. This is easily done by vectoring.

' INTERPRET

--- addr

MVP-FORTH

A user variable containing the compilation address to be executed by INTERPRET. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

HEX

24 USER 'INTERPRET

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
'INTERPRET @ 2+ NFA ID.
```

' LOAD

Type the name of the function currently assigned to `INTERPRET`, assuming it is not headerless.

Comment: By vectoring the INTERPRET function, it is possible to dynamically change the implementation of that function. Thus it is not necessary to recompile the source to make such changes.

```
'KEY          --- addr          MVP-FORTH
```

A user variable containing the compilation address to be executed by KEY. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

HEX

26 USER 'KEY

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

'KEY @ 2+ NFA ID.

Type the name of the function currently assigned to KEY, assuming it is not headerless.

Comment: By vectoring the KEY function, it is possible to dynamically change the implementation of that function. Thus it is not necessary to recompile the source to make such changes.

```
'LOAD      --- addr      MVP-FORTH
```

A user variable containing the compilation address to be executed by LOAD. (*Fig*) Not defined. (79S) Not defined. (83S) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

HEX

28 USER 'LOAD

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
'LOAD @ 2+ NFA ID.
```

Type the name of the function currently assigned to LOAD, assuming it is not headerless.

Comment: By vectoring the LOAD function, it is possible to dynamically change the implementation of that function. Thus it is not necessary to recompile the source to make such changes.

' NUMBER

--- addr

MVP-FORTH

A user variable containing the compilation address to be executed by NUMBER. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

HEX

2A USER 'NUMBER

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
'NUMBER @ 2+ NFA ID.
```

Type the name of the function currently assigned to NUMBER, assuming it is not headerless.

Comment: By vectoring the NUMBER function, it is possible to dynamically change the implementation of that function. Thus it is not necessary to recompile the source to make such changes.

' R/W

```
'PAGE      --- addr      MVP-FORTH
```

A user variable containing the compilation address to be executed by PAGE. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

HEX

2C USER 'PAGE

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

'PAGE @ 2+ NFA ID.

Type the name of the function currently assigned to PAGE, assuming it is not headerless.

Comment: By vectoring the PAGE function, it is possible to dynamically change the implementation of that function. Thus it is not necessary to recompile the source to make such changes.

```
'R/W      --- addr      MVP-FORTH
```

A user variable containing the compilation address to be executed by R/W. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

HEX

2E USER 'R/W

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

'R/W @ 2+ NFA ID.

Type the name of the function currently assigned to R/W, assuming it is not headerless.

Comment: By vectoring the R/W function, it is possible to dynamically change the implementation of that function. Thus it is not necessary to recompile the source to make such changes.

' S --- addr MVP-FORTH SUPPLEMENTAL

Place the address of the top of the stack on the top of the stack. (*Fig*) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

: 'S SP@ ;

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

'S U.

The address of the top of the stack is pushed onto the stack and then printed leaving the stack unchanged.

Comment: This ideogram is an alias for SP@ and is defined here only for the convenience of those working with *Starting Forth*. Usually it is more useful to use DEPTH.

' STREAM --- addr MVP-FORTH

Returns the address of the next character in the input stream. (*Fig*) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

'T&SCALC

Implementations:

MVP: 8088/86:

```
: 'STREAM BLK @ ?DUP
  IF BLOCK ELSE TIB @ THEN
  >IN @ + ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: ." 'STREAM C@ 22 =
  IF 1 >IN +!
  ELSE 22 STATE @
    IF COMPILE ." THEN
      WORD DUP C@ 1+ OVER + C@ 22 = NOT
      ?STREAM STATE @
      IF C@ 1+ ALLOT
      ELSE COUNT TYPE
      THEN
    THEN ; IMMEDIATE
```

This example is taken from the MVP-FORTH source code.

Comment: This ideogram provides a useful factor to return the current address of the input stream.

'T&SCALC

--- addr

MVP-FORTH DISK I-O

A user variable containing the compilation address to be executed by T&SCALC. (*Fig*) defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
HEX
30 USER 'T&SCALC
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

'T&SCALC @ 2+ NFA ID.

Type the name of the function currently assigned to T&SCALC, assuming it is not headerless.

Comment: By vectoring the T&SCALC function, it is possible to dynamically change the implementation of that function. Thus it is not necessary to recompile the source to make such changes.

'TIB --- a1 NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined (F83) Points to characters entered by user. (F-PC) Points to characters entered by user.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

VARIABLE 'TIB

F-PC:

Same as F83.

Example:

'TIB @ .

Get the address of this variable, fetch its contents and print it.

Comment: 'TIB serves the same function as TIB in fig-Forth and MVP-FORTH and differs from the value #TIB in F83 and F-PC. 'TIB is the pointer to a location. #TIB and >IN are counters.

'TITLE --- addr MVP-FORTH UTILITY

A variable holding the compilation address executed by TRIAD to place a message at the bottom of each printed page. (Fig) Not defined. (79S)

' VOCABULARY

Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
VARIABLE 'TITLE ' TITLE CFA 'TITLE !
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
FIND CR 'TITLE !
```

Set the vector so that TRIAD will print no message text at the bottom of the page.

Comment: By defining special message ideograms, any text may be inserted at the bottom of each page of copy. The code for a form-feed could be included with the title if desired. The ideogram TITLE supplies the default message.

' VOCABULARY

MVP-FORTH

A user variable containing the compilation address to be executed by VOCABULARY. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
HEX
```

```
32 USER 'VOCABULARY
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
'VOCABULARYFIG CFA 'VOCABULARY !
```

The example will change the function of VOCABULARY in MVP-FORTH from the 79S to fig-Forth.

Comment: The function of the ideogram, VOCABULARY, as defined in the 79S, is more restrictive than in fig-Forth. Although it is non-standard, many users prefer the latter function. Others, have new functions for VOCABULARY. Vectoring makes it possible to change functions easily.

'WORD --- addr MVP-FORTH

A user variable containing the compilation address to be executed by WORD. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Leaves the same address as WORD. In this system, 'WORD is the same as HERE. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

HEX

34 USER 'WORD

Fig: 8088/86:

Not used.

F83: 8088/86:

: 'WORD HERE ;

F-PC:

Not used.

Example:

```
'WORD @ 2+ NFA ID.
```

Type the name of the function currently assigned to WORD, assuming it is not headerless.

Comment: By vectoring the WORD function, it is possible to dynamically change the implementation of that function. Thus it is not necessary to recompile the source to make such changes.

(--- MVP-FORTH

Accept and ignore comment characters from the input stream, until the next right parenthesis. As a word, the left parenthesis must be

(

followed by one blank. It may be freely used while executing or compiling. An error condition exists if the input stream is exhausted before the right parenthesis. (*Fig*) Ignore a comment that will be delimited by a right parenthesis on the same line. May occur during execution or in a colon-definition. A blank after the leading parenthesis is required. (79S) Same as MVP. (83S) The characters ccc, delimited by) (closing parenthesis), are considered comments. Comments are not otherwise processed. The blank following (is not part of ccc. (may be freely used while interpreting or compiling. The number of characters in ccc may be from zero to the number of characters remaining in the input stream up to the closing parenthesis. (F83) The Forth Comment Character. The input stream is skipped until a) is encountered. (F-PC) Same as F83.

Pronounced: paren close-paren

Implementations:

MVP: 8088/86:

```
: (
  1 >IN +! 29 WORD C@ 1+ HERE +
  C@ 29 = NOT ?STREAM ;
  IMMEDIATE
```

Fig: 8088/86:

```
HEX
: ( 29 WORD ;
```

F83: 8088/86:

```
: ( ASCII ) PARSE 2DROP ; IMMEDIATE
```

F-PC:

Same as F83.

Example:

```
( THIS IS A COMMENT )
```

Upon executing this ideogram, the interpreter skips through the input stream until a close-paren (right parenthesis) is found. The enclosed comment is in effect, one big ignored ideogram.

Comment: This ideogram is common in all versions of FORTH. Customarily, the first line of every source screen is used for a comment, holding a screen title, author's initials, and the date of last modification. Unlike other languages, Forth's parentheses may not be nested.

(.) n1 --- a n2 NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined (F83) Convert a signed 16-bit number to a string. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

: (.)

DUP ABS 0 <# #S ROT SIGN #> ;

F-PC:

Same as F83.

Example:

1234 (.) TYPE

Enter the value followed by the ideogram (.) will leave the address and count on the stack for TYPE. This is equivalent to entering the value followed by the ideogram . (dot).

Comment: By leaving the address and count of the number string on the stack, the string can be used in various ways in text formatting.

(. ") --- C NOT USED IN MVP-FORTH

(Fig) The run-time procedure, compiled by ." which transmits the following in-line text to the selected output device. (79S) Not defined. (83S) Not defined. (F83) Type the inline string. Skip over it. (F-PC) Type the inline string, and continue execution after the string.

Implementations:

MVP: 8088/86:

The function of this ideogram has been assigned to
<.">.

Fig: 8088/86:

: (.") R COUNT DUP 1+
R> + >R TYPE ;

F83: 8088/86:

: (.") >R COUNT 2DUP + EVEN >R TYPE ;

(+LOOP)

F-PC:

```
DEFER (." )
: % (." )
  2R@ 2DUP C@L >R 1+ R@ TYPEL R>
  1+ XEVEN R> + >R ;
' % (." ) IS (." )
```

Example:

```
: ." COMPILE (." ) , " ; IMMEDIATE
```

The example is taken from the F83 source code.

Comment: This ideogram, because of the close-paren, plays havoc with Forth comments. Thus, <."> has been assigned its function in MVP-FORTH.

(+LOOP)

n ---

NOT USED IN MVP-FORTH

(Fig) The run-time procedure compiled by +LOOP, which increments the loop index by n and tests for loop completion. See +LOOP. (79S) Not defined. (83S) Not defined. (F83) Increment the loop counter by the value on the stack and decide whether or not to loop again. Due to the wierdness of the 8080, you have to stand on your head to determine the conditions under which you loop or exit. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

The function of this ideogram has been assigned to <+LOOP>.

Fig: 8088/86:

```
CODE (+LOOP)    BX POP    XLOO1 JMP
```

F83: 8088/86:

```
CODE (+LOOP)    AX POP    PLOOP #) JMP    END-CODE
```

F-PC:

```
CODE (+LOOP)
  AX POP    ADD 0 [RP], AX    OV<>
  IF    MOV EX: IP, 0 [IP]    NEXT
  THEN    ADD RP, # 6    ADD IP, # 2
  NEXT    END-CODE
```

Example:

```
: +LOOP
  COMPILE (+LOOP) 3000 ?PAIRS DUP 2+
```

<RESOLVE >RESOLVE ; IMMEDIATE

The example is taken from the F83 source code.

Comment: This ideogram, because of the close-paren, plays havoc with Forth comments. Thus, <+LOOP> has been assigned its function in MVP- FORTH.

(;CODE) --- NOT USED IN MVP-FORTH

Fig) The run-time procedure, compiled by ;CODE, that rewrites the code field of the most recently defined word to point to the following machine code sequence. (79S) Not defined. (83S) Not defined. (F83) Set the code field to the address of the following. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

The function of this ideogram has been assigned to
<;CODE>.

Fig: 8088/86:

```
: ( ;CODE ) R> LATEST PFA CFA ! ;
```

F83: 8088/86:

```
: ( ;CODE ) R> LAST @ NAME> ! ;
```

F-PC:

```
: ( ;CODE )
  2R @L LAST @ NAME>
  dup>r 232 ( CALL ) R@ C! \ Make a CALL not JUMP
  3 + - R> 1+ ! ;
```

Example:

```
: ;CODE
  ?CSP COMPILE ( ;CODE ) [COMPILE] [
  REVEAL ASSEMBLER ; IMMEDIATE
```

The example is taken from the F83 source code.

Comment: This ideogram, because of the close-paren, plays havoc with Forth comments. Thus, <;CODE> has been assigned its function in MVP- FORTH.

(?DO) last first --- NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) The runtime code compiled by ?DO. The difference between ?DO and DO is

(?LEAVE)

that ?DO will not perform any iterations if the initial index is equal to the final index. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE (?DO)

```
AX POP    BX POP    AX BX CMP    PDO JNE
0 [IP] IP MOV    NEXT
```

F-PC:

CODE (?DO)

```
POP DX    POP BX    CMP BX, DX    0=
IF MOV ES: IP, 0 [IP]    NEXT    THEN
XCHG RP, SP LODSW ES:    PUSH AX    ADD BX, # $8000
PUSH BX    SUB DX, BX    PUSH BX    XCHG RP, SP
NEXT    END-CODE
```

Example:

```
: ?DO    COMPILE    (?DO)    ?>MARK
```

The example is taken from the source code of F-PC

Comment: The ideogram is a primitive as illustrated.

(?LEAVE)

f ---

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Leaves if the flag on the stack is true. Continues if not. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE (?LEAVE)

```
AX POP    AX AX OR    PLEAVE JNE    NEXT
```

F-PC:

```
CODE (?LEAVE)
  POP AX      OR AX, AX      0=
  IF  NEXT    THEN
  MOV IP, 4 [RP]  ADD RP, # 6
  NEXT      END-CODE
```

Example:

```
: ?LEAVE  COMPILE (?LEAVE) ;  IMMEDIATE
```

The example is take from the F-PC source code.

Comment: The ideogram is an F-PC primitive as illustrated.

(ABORT) --- NOT USED IN MVP-FORTH

(Fig) Executes after an error when WARNING is -1. This ideogram normally executes ABORT, but may be altered (with care) to a user's alternative procedure. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

The function of this ideogram has been dropped in MVP-FORTH.

Fig: 8088/86:

```
: (ABORT)  ABORT ;
```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
( Not used )
```

Not used in current systems.

Comment: This ideogram, because of the close-paren, plays havoc with Forth comments. Thus, <ABORT> has been assigned this function and is vectored from ABORT through the variable 'ABORT.

(D.) d --- a l NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Convert a signed double number to a string. (F-PC) Same as F83.

(DO)

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

: (D.)

TUCK DABS <# #S ROT SIGN #> ;

F-PC:

Same as F83.

Example:

-1234. (D.) TYPE

Enter a double precision negative value, format its output and type it.

Comment: A function for formatting integer values in preparation for printing them.

(DO)

NOT USED IN MVP-FORTH

(Fig) The run-time procedure compiled by DO which moves the loop control parameters to the return stack. See DO. (79S) Not defined. (83S) Not defined. (F83) The run time code compiled by DO. Pushes the inline address onto the return stack along with values needed by (LOOP). (F-PC) Same as F83

Implementations:

MVP: 8088/86:

The function of this ideogram has been assigned to <DO>.

Fig: 8088/86:

CODE (DO)

DX POP AX POP SP BP, XCHG

AX PUSH DX PUSH SP BP, XCHG NEXT JMP

F83: 8088/86:

CODE (DO)

AX POP BX POP

LABEL PDO RP DEC RP DEC 0 [IP] DX MOV

DX SS: 0 [RP] MOV IP INC IP INC

8000 # BX ADD RP DEC RP DEC

```

BX SS: 0 [RP] MOV    BX AX SUB    RP DEC    RP DEC
AX SS: 0 [RP] MOV    NEXT

```

F-PC:

```

CODE (DO)
  POP DX    POP BX
  XCHG RP, SP    \ 4
  LODSW ES:    \ 12 + 2
  PUSH AX      \ 11
  ADD BX, # $8000 \ 4
  PUSH BX      \ 11
  SUB DX, BX    \ 3
  PUSH DX      \ 11
  XCHG RP, SP    \ 4    = 62
  NEXT    END-CODE

```

Example:

```

: DO
  COMPILE (DO)    >MARK 3000 ; IMMEDIATE

```

The example is taken from the F83 source code.

Comment: This ideogram, because of the close-paren, plays havoc with Forth comments. Thus, <DO> has been assigned its function in MVP-FORTH.

(FIND)

NOT USED IN MVP-FORTH

```

addr1 addr2 --- pfa S tf    (ok)
addr1 addr2 --- ff          (bad)

```

(Fig) Searches the dictionary starting at the name field address addr2, matching to the text at addr1. Returns parameter field address, length byte of name field and boolean true for a good match. If no match is found, only a boolean false is left. (79S) Not defined. (83S) Not defined. (F83) Does a search of the dictionary based on a pointer to a vocabulary thread and a string. If it finds the string in the chain, it returns a pointer to the CFA field inside the header. This field contains the code field address of the body. If it was an immediate word the flag returned is a 1. If it is non-immediate the flag returned is a -1. If the name was not found, the string address is returned along with a flag of zero. Note that links point to links, and are absolute addresses. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

(FIND)

The function of this ideogram has been assigned to <FIND>.

Fig: 8088/86:

```
HEX
CODE (FIND)
  DS AX, MOV  AX ES, MOV  BX POP  CX POP
HERE LABEL AFIND1  CX DI, MOV  [BX] AL, MOV
  AL DL, MOV  [DI] AL, XOR   3F AL, AND  AFIND3 JNZ
HERE LABEL AFIND2  BX INC  DI INC  [BX] AL, MOV
  [DI] AL, XOR  AL AL, ADD  AFIND3 JNZ
  AFIND2 JNB   5 BX, ADD  BX PUSH  1
  1 AX,MOV  DH DH, SUB  DPUSH JMP
HERE LABEL AFIND3  BX INC  AFIND4 JB  [BX] AL, MOV
  AL AL, ADD  AFIND3 JMP
HERE LABEL AFIND4  [BX] BX, MOV  BX BX, OR
  AFIND1 JNZ   0 AX MOV  APUSH JMP
```

F83: 8088/86:

```
CODE (FIND)
  BX POP  BX BX OR  0=
  IF  AX AX SUB  1PUSH,  THEN
  SI DX MOV  HSEG #) AX MOV  AX ES MOV
BEGIN  BX DI MOV ( link )  6 # DI ADD ( name )
  SI POP  ( here )  SI PUSH  AL LODS ( len )
  AL CL MOV  ES: 0 [DI] AL XOR  63 # AL AND  0=
  IF  DI INC  CH CH SUB  REPZ BYTE CMPS  0=
    IF  SI POP  ES: 4 [BX] PUSH
      ES: 6 [BX] AL MOV  64 # AL AND  0<>
      IF  1 # AX MOV  ELSE  -1 # AX MOV  THEN
      DX SI MOV  1PUSH,
      THEN THEN  ES: 0 [BX] BX MOV  BX BX OR  0=
  UNTIL  AX AX SUB  DX SI MOV  1PUSH
```

F-PC:

```
CODE (FIND)
  POP BX  OR BX, BX  0=
  IF  SUB AX, AX  1PUSH  THEN
  POP CX  PUSH ES  MOV ES, YSEG  MOV DI, CX
BEGIN  MOV ES: AX, 2 [BX]  XOR AX, 0 [DI]
  AND AX, # ( 63 ) $7F3F  0=
  IF  MOV DX, BX  ADD BX, # 2
  BEGIN  INC BX  INC DI
    MOV ES: AL, 0 [BX]  XOR AL, 0 [DI]  0<>
```



```
UNTIL   AND AL, # 127  0=
IF      MOV ES: CX, 1 [BX]  \ pick up CFA
      MOV BX, DX   MOV ES: AL, 2 [BX]
      AND AL, # 64    0<>
      IF      MOV AX, # 1
      ELSE    MOV AX, # -1
      THEN    POP ES   PUSH CX   1PUSH
      THEN    MOV BX, DX   MOV DI, CX
      THEN    MOV ES: BX, 0 [BX]   OR BX, BX   0=
UNTIL      POP ES   PUSH CX   SUB AX, AX
1PUSH      END-CODE
```

Example:

```
: FIND
DUP C@
IF FALSE #VOCS @
  DO DROP CONTEXT I 2* + @ DUP
  IF OVER 1+ @ SWAP HASH @ (FIND) DUP ?LEAVE THEN
  LOOP
ELSE DROP [' ] NOOP 1
THEN ;
```

The example is taken from the F83 source code.

Comment: This ideogram, because of the close-paren, plays havoc with Forth comments. Thus, <FIND> has been assigned its function in MVP-FORTH.

(LINE)**NOT USED IN MVP-FORTH**

n1 n2 --- addr count

(Fig) Convert the line number n1 and the screen n2 to the disc buffer address containing the data. A count of 64 indicates the full line text length. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

The function of this ideogram has been assigned to <LINE>.

Fig: 8088/86:

```
: (LINE)  >R  64  B/BLK  */MOD  R>  B/SCR  *  +
BLOCK  +  64  ;
```

(LOOP)

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: .LINE
  (LINE)  -TRAILING  TYPE  ;
```

The example is taken from the Fig: 8088/86 source code.

Comment: This ideogram, because of the close-paren, plays havoc with Forth comments. Thus, <LINE> has been assigned its function in MVP-FORTH.

(LOOP)

NOT USED IN MVP-FORTH

(Fig) The run-time procedure compiled by LOOP which increments the loop index and tests for loop completion. See LOOP. (79S) Not defined. (83S) Not defined. (F83) The runtime procedure for LOOP. Branches back to the beginning of the loop if there are more iterations to do. Otherwise it exits. The loop counter is incremented. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

The function of this ideogram has been assigned to
<LOOP>

Fig: 8088/86:

```
CODE (LOOP)
  1 BX, MOV
  HERE LABEL XLOO1  BX [BX], ADD
    [BP] AX, MOV  2[BP] AX, SUB  BX AX, XOR
    BRAN1 JS  4 BP, ADD  SI INC  SI INC  NEXT JMP
```

F83: 8088/86:

```
HEX
CODE (LOOP)
  1 # AX MOV
  LABEL PLOOP  AX SS: 0 [RP] ADD  BRAN1 JNO
    6 # RP ADD  IP INC  IP INC  NEXT
DECIMAL
```

F-PC:

```
CODE (LOOP)
  INC 0 [RP] WORD  OV<>
  IF  MOV ES: IP, 0 [IP]  NEXT
  THEN  ADD RP, # 6  ADD IP, # 2
  NEXT  END-CODE
```

Example:

```
: LOOP
  COMPILE (LOOP) 3000 ?PAIRS
  DUP 2+
  <RESOLVE >RESOLVE ; IMMEDIATE
```

The example is taken from the F83 source code.

Comment: This ideogram, because of the close-paren, plays havoc with Forth comments. Thus, <LOOP> has been assigned its function in MVP-FORTH.

(NUMBER)

NOT USED IN MVP-FORTH

n1 n2 --- addr count

(Fig) Convert the ASCII text beginning at addr1+1 with regard to BASE. The new value is accumulated into double number d1, being left as d2. addr2 is the address of the first unconvertable digit. Used by NUMBER. (79S) Not defined. (83S) Not defined. (F83) Convert the count delimited string at addr to a double number. (NUMBER) takes into account a leading minus sign, and stores a pointer to the last period in DPL. Note the string must end with a blank or an error message is issued. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

This ideogram is not implemented in MVP-FORTH.

Fig: 8088/86:

```
: (NUMBER)
  BEGIN 1+ DUP >R C@ BASE @ DIGIT
  WHILE SWAP BASE @ U* DROP ROT BASE @ U* D+ DPL @ 1+
    IF 1 DPL +! ENDIF R>
  REPEAT R> ;
```

F83: 8088/86:

```
: (NUMBER)  NUMBER? NOT ?MISSING ;
```

F-PC:

Same as F83.@EXAMPLE H = Example:

(U.)

Example:

```
' (NUMBER) IS NUMBER
```

The example is taken from the F83 source code.

Comment: This ideogram is obsolete and has been completely dropped. The 79S now uses CONVERT for its function. Note that *Starting Forth* also allows the use of BINARY.

Though the ideogram is also absent from the 79S, both F83 and F-PC continue to include it.

(S --- NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: (S ( ; IMMEDIATE
```

F-PC:

Not used.

Example:

```
(S addr1 addr2 --- )
```

The example illustrates the way the ideogram is commonly used.

Comment: The ideogram is commonly used to suggest that the enclosed comment represents a stack image before and after a function. It can be implemented as an alias for (.

(U.) u --- a l NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Convert an unsigned 16-bit number to a string. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

: (U.)

0 <# #S #> ;

F-PC:

Same as F83.

Example:

123 (U.) TYPE

The example performs the function of U. .

Comment: In some Forth implementations this ideogram is a factor of U. .

* n1 n2 --- n3

MVP-FORTH

Leave the arithmetic product of n1 times n2. (*Fig*) Leave the signed product of two signed numbers. (.79S) Same as MVP. (83S) n3 is the least-significant 16 bits of the arithmetic product of n1 time n2. (F83) Not defined. (F-PC) Form a 16-bit product from two 16-bit numbers.

Pronounced: times

Implementations:

MVP: 8088/86:

: * U* DROP ;

Fig: 8088/86:

: * M* DROP ;

F83: 8088/86:

: * UM* DROP ;

F-PC:

CODE *

POP AX POP BX MUL BX

1PUSH END-CODE

Example:

3 -6 * .

The two single precision signed integers are placed on the stack, and then multiplied together leaving a single precision signed value on the stack which is then printed.

Comment: Should an overflow occur, it will go undetected. The result

cases, its 32-bit intermediate precision and use of ratios eliminate the need for floating point arithmetic. Be careful of the floored division in the 83S.

*** /MOD** n1 n2 n3 --- n4 n5 MVP-FORTH

Multiply n1 by n2, divide the result by n3 and leave the remainder n4 and quotient n5. A 32-bit intermediate product is used as for ***/**. The remainder has the same sign as n1. (*Fig*) Leave the quotient n5 and the remainder n4 of the operation $n1 * n2 / n3$. A 31 bit intermediate product is used as for ***/**. (*79S*) Same as MVP. (*83S*) n1 is first multiplied by n2 producing an intermediate 32-bit result. n4 is the remainder and n5 is the floor of the quotient of the intermediate 32-bit result divided by the divisor n3. A 32-bit intermediate product is used as for ***/**. n4 has the same sign as n3 or is zero. An error condition results if the divisor is zero or if the quotient falls outside of the range $\{-32,768..32,767\}$. See: division, floored. (*F83*) Not defined. (*F-PC*) This returns the floored quotient and modulus of a 32-bit numerator and a 16-bit denominator n3. The numerator is an intermediate 32-bit product of the 16-bit quantities n1 and n2. Note that the sign of the modulus is the same as the sign of the denominator.

Pronounced: times-divide-mod

Implementations:

MVP: 8088/86:

```
: */MOD >R M* R> M/ ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
: */MOD >R *D R> M/MOD ;
```

F-PC:

```
CODE */MOD
```

```
POP BX POP AX POP CX
IMUL CX MOV CX, BX XOR CX, DX 0>=
IF IDIV BX 2PUSH
THEN IDIV BX OR DX, DX 0<>
IF ADD DX, BX DEC AX
THEN 2PUSH END-CODE
```

Example:

```
13 5 3 */MOD . .
```


is added to the n value at addr using the convention for +. This sum replaces the original value at addr. (F83) Increment the value at addr by n. This is equivalent to the following: DUP @ ROT + SWAP !, but is much faster. (F-PC) Same as F83.

Pronounced: plus-store

MVP: 8088/86:

CODE +!

BX POP AX POP AX [BX], ADD NEXT JMP END-CODE

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

CODE +!

BX POP AX POP AX 0 [BX] ADD NEXT

F-PC:

Same as F83.

Example: CAUTION: May corrupt or crash your system.

HEX 1 4CD2 +! DECIMAL

Increments the 16-bit cell at memory location 4CD2 by 1.

Comment: Particularly useful for incrementing and decrementing counters in memory. Carry and overflow are ignored. Note: Since this ideogram will write to any location in machine address space, take care not to corrupt your dictionary, nucleus, or operating system.

+- n1 n2 --- n3

MVP-FORTH

Apply the sign of n2 to n1, which is left as n3. (Fig) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

: +- 0< IF NEGATE THEN :

Fig: 8088/86:

: +- 0< IF MINUS ENDIF ;

F83: 8088/86:

Not used.

F-PC:

Not used.

+BUF

Example:

-4 -2 +- .

Since the -2 on top is less than zero, the -4 underneath is negated, leaving the result of 4, which is printed.

Comment: Simplifies the implementation of some signed multiplication and division operations.

+BUF addr1 --- addr2 f MVP-FORTH

Advance the disc buffer address addr1 to the address of the next buffer addr2. Boolean f is false when addr2 is the buffer presently pointed to by the variable PREV. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: +BUF    HDBT   ( 404H ) +   DUP   LIMIT   =  
         IF DROP FIRST THEN  
         DUP PREV @ - ;
```

Fig: 8088/86:

```
HEX  
: +BUF    84 ( System dependent ) +   DUP   LIMIT   =  
         IF DROP FIRST DUP PREV @ - ;
```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: BUFFER    USE @   DUP R  
         BEGIN +BUF UNTIL  
         USE ! R@ @ 0  
         IF R@ 2+ R@ @ 7FFF AND 0 R/W THEN  
         R@ ! R@ PREV ! R> 2+ ;
```

This example is taken from the system source code. Note that +BUF is cyclic; when its value reaches LIMIT, it short-circuits around to the start of the buffer area.

Comment: This ideogram, used in the implementation of the 79S ideograms, BLOCK and BUFFER, may be available to the programmer. The early fig-Forth implementation contains a bug which affects FLUSH

in BLOCK and BUFFER
+LOOP

n ---

MVP-FORTH

Add the signed increment *n* to the loop index using the convention for +, and compare the total to the limit. Return execution to the corresponding DO until the new index is equal to or greater than the limit ($n > 0$) or until the new index is less than the limit ($n < 0$). Upon the exiting from the loop, discard the loop control parameters, continuing execution ahead. Index and limit are signed integers in the range -32768 ... 32767. (*Fig*) At run-time, +LOOP selectively controls branching back to the corresponding DO based on *n1*, the loop index and the loop limit. The signed increment *n1* is added to the index and the total compared to the limit. The branch back to DO occurs until the new index is equal to or less than the limit ($n1 > 0$), or until the new index is equal to or less than the limit ($n1 < 0$). Upon exiting the loop, the parameters are discarded and execution continues ahead. At compile time, +LOOP compiles the run-time word (+LOOP) and the branch offset computed from HERE to the address left on the stack by DO. *n2* is used for compile time error checking. (*Fig 8088/86 Modification:*) The run-time portion of this word (LOOP) was modified to conform to Forth-79 standards when dealing with an "index" that is less than zero (negative). The loop will stop when the "index" is less than the "limit" (the fig model will stop when less than or equal to). (79S) Same as MVP. (83S) *n* is added to the loop index. If the new index was incremented across the boundary between limit-1 and limit then the loop is terminated and loop control parameters are discarded. When the loop is not terminated, execution continues to just after the corresponding DO. sys is balanced with its corresponding DO. See DO. (F83) Not defined. (F-PC) Terminate a loop structure. Increment loop index by *n* and repeat <loop-body> until loop index crosses the boundary between limit and limit - 1. Used in the form DO <loop-body> LOOP.

(*Comment:* It is a historical precedent that the limit for *n* is irregular. Further consideration of the characteristic is likely.)

Pronounced: plus-loop

Implementations:

MVP: 8088/86:

: +LOOP

3 ?PAIRS COMPILE <+LOOP> HERE - , ;

Fig: 8088/86:

+ORIGIN

```
: +LOOP
  3 ?PAIRS COMPILE (+LOOP) BACK ;
```

F83: 8088/86:

```
: +LOOP
  COMPILE (+LOOP) 3000 ?PAIRS DUP 2+
  <RESOLVE >RESOLVE ; IMMEDIATE
```

F-PC:

```
: +LOOP
  COMPILE (+LOOP) 2DUP 2+ ?<RESOLVE
  ?>RESOLVE ; IMMEDIATE
```

Example:

```
: TEST 10 1 DO I . 3 +LOOP ;
```

Executing TEST will cause the values 1 4 7 to be printed.

Comment: This ideogram works with DO to form a nestable control structure LEAVE may be used to terminate the loop before the index has run its full course. Various Forth implementations react differently to an index which changes sign over its range.

Note that when the value of n1 is less than zero the loop is decreasing which means that the first value before the DO is less than the second. Also note that in such a decreasing loop the loop will be executed when the index is equal to the limit. This is different from the usual ascending loop which terminates when the index equals the limit. Furthermore, because of the signed values, the index cannot be used as an address when crossing the extremes of signed numbers.

+ORIGIN n --- addr NOT USED IN MVP-FORTH

(Fig) Leave the memory address relative by n to the origin parameter area. n is the minimum address unit, either byte or word. This definition is used to access or modify the boot-up parameters at the origin area. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86: Could be defined as:

```
: +ORIGIN ORIGIN + ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Not used.

F-PC:

Not used.

(This implementation would only be applicable in fig-Forth. It is of no use in MVP-FORTH.)

Example:

```
VOC-LINK @ 20 +ORIGIN !
```

This example would only apply to a fig-Forth implementation. The current value in the user variable VOC-LINK is fetched and stored in its corresponding location in the bootup parameter area of memory.

Comment: During a cold start, the user variables such as VOC-LINK must be initialized. Their initial values are fetched from an area of "boot up parameters", located near the bottom of the system at a fixed offset from the origin. Since the origin may vary among implementations, the proper absolute addresses are calculated from an offset and a base location. Instead of +ORIGIN, MVP-FORTH uses the constant INIT-USER to locate the start of this area.

, n ---

MVP-FORTH

Allot two bytes in the dictionary, storing n there. (*Fig*) Store n into the next available dictionary memory cell, advancing the dictionary pointer. (*79S*) Same as MVP. (*83S*) ALLOT space for 16b then store 16b at HERE 2-. (*F83*) Set the contents of the dictionary value on the stack. (*F-PC*) Set the contents of the dictionary to the arbitrary 16-bit value on the stack.

Pronounced: comma

Implementations:

MVP: 8088/86:

```
: ,    HERE ! 2 ALLOT ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Same as MVP.

F-PC:

```
CODE ,
```

```
MOV BX, UP    MOV AX, DP [BX]
```

```
ADD DP [BX], # 2 WORD    MOV BX, AX
```

POP CX MOV 0 [BX], CX
NEXT END-CODE

Example:

```
: LITERAL STATE @  
IF COMPILE LIT , THEN ; IMMEDIATE
```

This example from the MVP-FORTH source code uses comma to place a literal value in-line with compiled code.

Comment: This ideogram is useful for initializing arrays of integers at compile time.

, " --- NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined (F83) Add the following text till a " to the dictionary. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: , "  
ASCII " PARSE TUCK 'WORD PLACE 1+ ALLOT ALIGN ;
```

F-PC:

```
: , "  
ASCII " PARSE TUCK 'WORD PLACE 1+ ALLOT ;
```

Example:

```
HERE DUP 20 DUMP  
, " Now is the time. "  
20 DUMP
```

Illustrate the function by doing a DUMP from HERE before and after using the ideogram to insert a count byte and a string.

Comment: The ideogram is in common usage and has been added to F83 and F-PC. No functional definition is included.

— n1 n2 --- n3 MVP-FORTH

Subtract n2 from n1 and leave the difference n3. (Fig) Leave the

difference of n1-n2. (79S) Same as MVP. (83S) n3 is the result of subtracting n2 from n1 (F83) Subtracts n2 from n1 leaving the result on the stack. (F-PC) Same as F83.

Pronounced: minus

Implementations:

MVP: 8088/86:

```
CODE -
  DX POP    AX POP    DX AX, SUB
  APUSH JMP  END-CODE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE -
  BX POP    AX POP    BX AX SUB    1PUSH
```

F-PC:

Same as F83.

Example:

23 -12 - .

In this example, -12 is subtracted from 23, yielding the result, 35 , which is printed.

Comment: A basic arithmetic operator. As usual, borrow and overflow conditions are ignored.

-->

NOT USED IN MVP-FORTH

(Fig) Continue interpretation with the next disk screen. (79S *Reference Word Set*) Same as MVP. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Pronounced: next-block

Implementations:

MVP: 8088/86: Could be defined as:

```
: -->    BLK @ .    ?LOADING 0 >IN !    1 BLK +! ;
```

Fig: 8088/86:

```
: -->    ?LOADING 0 >IN !    B/SCR BLK @
      OVER MOD - BLK +! ;
```

F83: 8088/86:

-DUP

: --> >IN OFF 1 BLK +! ; IMMEDIATE

F-PC:

Not used.

Example:

(On line 8 of Screen # 102:)

-->

Assume the ideogram appears on line 8 of screen 102: Loading of screen 102 proceeds normally until it is encountered. The input stream is then diverted to the start of screen 103, bypassing lines 9-15 of screen 102.

Comment: This is one of several ways of loading a series of screens. In any case one should not end a screen with the next screen number followed by LOAD. This can produce a heavy load on the return stack. By using -->, the input stream from that screen is terminated and started at the beginning of the next sequential screen. It has a difficulty when a series of screens being loaded needs to be interrupted to insert an additional screen. The interrupting screen must be edited first. It is better to use THRU, or better still, a load screen with a list which can be commented.

-DUP

NOT USED IN MVP-FORTH

```
      n --- n      (if zero)
      n --- n n    (if non-zero)
```

(Fig) Reproduce n only if it is non-zero. This is usually used to copy a value just before IF, to eliminate the need for an ELSE part to drop it. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86: Could be defined as:

```
: -DUP    ?DUP ;
```

Fig: 8088/86:

```
: -DUP    DUP
  IF DUP DENDIF ;
```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: TEST    -DUP  IF  ." NON-ZERO RETURN CODE: " .  
    THEN ;
```

Frequently, as in the above example, a non-zero value will require some sort of processing, while no action will be needed for zero. In this kind of situation, using -DUP or better ?DUP, in front of IF will save coding the additional clause "ELSE DROP".

Comment: This ideogram is obsolete, being replaced by 79S's ?DUP. Although -DUP may be included for fig-Forth compatibility, its use is discouraged.

-FIND

MVP-FORTH

```
--- pfa  b  tf      (if found)  
--- ff      (if not found)
```

Accepts the next word (delimited by blanks) in the input stream to HERE and searches the CONTEXT and then the Forth vocabularies for a matching entry. If found, the dictionary entry's parameter field address, its length byte, and a boolean true is left. Otherwise, only a boolean false is left. (*Fig*) Same as MVP. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: -FIND  '-FIND  @  EXECUTE  ;
```

Fig: 8088/86:

```
: -FIND  
  BL WORD HERE CONTEXT @ @ (FIND) DUP 0=  
  IF DROP HERE LATEST (FIND) ENDIF ;
```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
-FIND  JUNK
```

Search the dictionary for the ideogram JUNK. Presumably, the word is not in the dictionary and the flag value of 0 is left on the stack.

Comment: Both -FIND and FIND are used. The difference is that -FIND

-TEXT

leaves a parameter field address and length byte, while FIND leaves only a code field address on the stack. They are otherwise the same.

Note: the null character used to terminate the terminal and disk buffers is defined as an ideogram in the dictionary (See x). This may occasionally produce bewildering error messages or unexpected results when you use the ideograms: -FIND, ' , [COMPILE], FORGET, or others which search the dictionary or define new words. In MVP- FORTH, -FIND is vectored to <-FIND>.

-ROT

NOT USED IN MVP-FORTH

n1 n2 n3 --- n3 n1 n2

(Fig) Not defined. (79S) Not defined. (83S) Not defined (F83) The inverse of ROT. Rotates the top element to third place. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE -ROT

BX POP AX POP DX POP BX PUSH 2PUSH

F-PC:

Same as F83.

Example:

1 2 3 -ROT .S

Place the three values on the stack, execute the function and then display the stack.

Comment: The ideogram adds to the stack operations.

-TEXT

MVP-FORTH SUPPLEMENTAL

addr1 n1 addr2 --- n2

Compare two strings over the length n1 beginning at addr1 and addr2. Return zero if the strings are equal. If unequal, return n2, the difference between the last character compared: addr1(i) - addr2(i). (Fig) Not defined. (79S Reference Word Set) Same as MVP. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Pronounced: dash-text

Implementations:

MVP: 8088/86:

```
: -TEXT
  DDUP + SWAP
  DO DROP 2+ DUP 2- @ I @ - DUP
    IF DUP ABS / LEAVE THEN 2
  /LOOP SWAP DROP ;
```

Fig: 8088/86

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
HEX 4EC2 6 100 BLOCK -TEXT DECIMAL
```

See if the first six bytes beginning at memory address 4EC2 compare with the first six bytes in BLOCK number 100H which is brought into a memory buffer placing its beginning address on the stack. A flag is left on the stack according to the test.

Comment: An ideogram which appears in slightly different forms in many editors. This version compares two byte pairs at a time. Thus, the length *n1* must be even. A byte oriented version could be:

MVP: 8088/86:

```
: -TEXT
  DDUP + SWAP
  DO DROP 1+ DUP 1- C@ I C@ - DUP
    IF DUP ABS / LEAVE THEN 1
  /LOOP SWAP DROP ;
```

-TRAILING

MVP-FORTH

addr n1 --- addr n2

Adjust the character count *n1* of a text string beginning at *addr* to exclude trailing blanks, i.e., the characters at the *addr+n2* to *addr+n1-1* are blanks. An error condition exists if *n1* is negative. (*Fig*) Adjusts the character count *n1* of a text string beginning address to suppress the output of trailing blanks, i.e. the characters at *addr+n1* to *addr+n2* are

-TRAILING

blanks. (79S) Same as MVP. (83S) The character count +n1 of a text string beginning at addr is adjusted to exclude trailing spaces. If +n1 is zero, then +n2 is also zero. If the entire string consists of spaces, then +n2 is zero. (F83) Return the address and length of the given string ignoring trailing blanks. (F-PC) Same as F83.

Pronounced: dash-trailing

Implementations:

MVP: 8088/86:

```
: -TRAILING
  DUP 0
  DO DDUP + 1- C@ BL -
    IF LEAVE ELSE 1- THEN
  LOOP ;
```

Fig: 8088/86:

```
: -TRAINLING
  DUP 0
  DO OVER OVER + 1 - C@ BL -
    IF LEAVE ELSE 1 - ENDIF
  LOOP ;
```

F83: 8088/86:

```
CODE -TRAILING
  DS PUSH  ES POP  CX POP  DI POP  DI PUSH
  CX DI ADD  32 # AX MOV  DI DEC  STD
  REPZ BYTE SCAS  CLD  0<> IF  CX INC  THEN  CX PUSH
  NEXT
```

F-PC:

```
CODE -TRAILING
  POP CS  POP DI  PUSH DI  CX<>)
  IF  MOV AX, DS  PUSH ES  STD  MOV ES, AX
    ADD DI, CX  DEC DI  MOV AL, # $20
  REPE SCASB  0<>
  IF  INC CX  THEN
    CLD  POP ES
  THEN  PUSH CX  NEXT  END-CODE
```

Example:

```
PAD  COUNT  -TRAILING  TYPE
```

Print the text beginning at PAD plus 1 for the count at the byte whose address is PAD, but drop all trailing spaces from the length.

Comment: This ideogram saves time in typing output, but if some spacing is necessary for formatting it should not be used.

• n --- MVP-FORTH

Display n converted according to BASE in a free-field format with one trailing blank. Display only a negative sign. (*Fig*) Print a number from a signed 16-bit two's complement value, converted according to the numeric BASE. A trailing blanks follows. (*79S*) Same as MVP. (*83S*) The absolute value of n is displayed in a free field format with a leading minus sign if n is negative. (*F83*) Output as a signed single number with a trailing space. (*F-PC*) Output as a signed single number with a trailing space.

Pronounced: dot

Implementations:

MVP: 8088/86:

: . S->D D. ;

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

: . (.) TYPE SPACE ;

F-PC:

Same as F83.

Example:

43 .

Place the value 43 on the stack and then print it.

Comment: Printing a value removes it from the stack. In most implementations, a stack underflow check is not performed until after a number is printed, in which case the number is garbage.

• " --- MVP-FORTH

Interpreted or used in a colon-definition. Accept the following text from the input stream, terminated by " (double-quote). If executing, transmit this text to the selected output device. If compiling, compile so that later execution will transmit the text to the selected output device. At least 127 characters are allowed in the text. If the input stream is exhausted before the terminating double-quote, an error condition exists. (*Fig*) Compile an in-line string cccc (delimited by the trailing

. "

") with an execution procedure to transmit the text to the selected output device. If executed outside a definition, ." will immediately print the text until the final ". The maximum number of characters may be an installation dependent value. See (.). (79S) Same as MVP. (83S) Later execution will display the characters ccc up to but not including the delimiting ' (close-quote). The blank following ." is not part of ccc. (F83) Compile the string to be typed out later. (F-PC) Compile the string till a " to be typed out later. These strings are compiled into LIST space. It is also displayed from LIST space.

Form: ." cccc"

Pronounced: dot-quote

Implementations:

MVP: 8088/86:

```
: ."
  'STREAM C@ 22 =
  IF 1 >IN +!
  ELSE 22 STATE @
    IF COMPILE <."> THEN
      WORD DUP C@ 1+ OVER + C@
      22 = NOT ?STREAM STATE @
      IF C@ 1+ ALLOT
        ELSE COUNT TYPE
        THEN
      THEN ; IMMEDIATE
```

Fig: 8088/86:

```
HEX
: ."
  22 STATE @
  IF COMPILE (".") WORD HERE C@ 1+ ALLOT
  ELSE WORD HERE COUNT TYPE
  ENDIF ; IMMEDIATE
```

F83: 8088/86:

```
: ." COMPILE (".") , " ; IMMEDIATE
```

F-PC:

```
: ." COMPILE (".") X, " ; IMMEDIATE
```

Example:

```
." PRINT THIS"
```

Entering this source will cause the contents between the double quotes

Comment: This ideogram generally does the same thing among the various versions of Forth. However, not all versions will print one or more blank spaces, and in some versions the buffer size may be different.

Comment: This ideogram generally does the same thing among the various versions of Forth. However, not all versions will print one or more blank spaces, and in some versions the buffer size may be different.

```

. (      ---      NOT USED IN MVP-FORTH

```

(Fig) Not defined. (79S) Not defined. (83S) Used in the form: . (ccc). The characters ccc up to but not including the delimiting) (closing parenthesis) are displayed. The blank following ." is not part of ccc. (F83) Type the following string on the terminal. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: .(
  ASCII ) PARSE >TYPE ; IMMEDIATE
```

F-PC:

Same as F83.

Example:

```
.( Hello there! )
```

The example will display the phrase on the screen.

Comment: The ideogram differs from `."` in that the string cannot be compiled into a definition.

```
.ID      addr ---      NOT USED IN MVP-FORTH
```

(Fig) Not defined. **(79S)** Not defined. **(83S)** Not defined. **(F83)** Display the variable length name whose name field address is on the stack. If it is shorter than its count, it is padded with underscores. Only valid ASCII is typed. **(F-PC)** Same as F83.

Implementations:

MVP: 8088/86:

Not used.

. INDEX

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: .ID
  DUP H@ 31 AND 0
  ?DO 1+ DUP H@ 255 AND EMIT LOOP DROP SPACE
;
```

F-PC:

```
DEFER .ID
' %ID IS .ID
```

Example:

```
' DUMP NAME .ID
```

The example finds the name field address and prints it.

Comment: Earlier implementations of Forth use the ideogram ID. .

. INDEX

n ---

MVP-FORTH UTILITY

Print line 0 on screen n. (*Fig*) Not defined. (*79S*) Not defined. (*83S*)
Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: .INDEX
  USE @ SWAP PAD USE ! DUP
  SPBLK * T&SCALC SEC-READ CR 4 .R 2 SPACES
  PAD C/L -TRAILING TYPE USE ! ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
20 .INDEX
```

Print line zero of screen 20.

Comment: This ideogram does the work inside INDEX. Since it bypasses Forth's buffer management and reads only the first sector of each

screen, it runs significantly faster than conventional versions of INDEX. This illustrates how nontransportable, nonstandard programs can sometimes run circles around their 79S equivalents.

.LINE line scr --- MVP-FORTH

Print on the terminal device, a line of text from the disk by its line and screen number. Trailing blanks are suppressed. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: .LINE <LINE> -TRAILING TYPE ;
```

Fig: 8088/86:

```
: .LINE (LINE) -TRAILING TYPE ;
```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
10 12 .LINE
```

Print the contents of line number 10 on screen 12.

Comment: A way to print any line on any screen. Using this ideogram, LOAD could be modified to list the 0 line of each screen, along with the screen number, before it is loaded to indicate the progress in loading a long series of screens and as one way to indicate the location of an error during loading.

.R n1 n2 --- MVP-FORTH

Print n1 right aligned in a field of n2 characters, according to BASE. If n2 is less than 1, no leading blanks are supplied. (*Fig*) Print the number n1 right aligned in a field whose width is n2. No following blank is printed. (79S *Reference Word Set*) Same as MVP. (83S) Not defined. (F83) Output as a signed single number right justified. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

.S

```
: .R  >R  S-D  R>  D.R  ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
: .R  >R  (.)  R>  OVER - SPACES  TYPE  ;
```

F-PC:

Same as F83.

Example:

```
3456  10  .R
```

Print the value 3456 right justified within a field of 10 spaces.

Comment: Although not available in all versions of Forth, this ideogram is easy to implement and quite useful in formatted output.

.S ---

MVP-FORTH UTILITY

.SL

.SR

.SS

These ideograms work in concert to implement a nondestructive stack display. .S will print the values on the stack in ascending or descending order, according to the flag in the constant .SS. The flag is set by .SL and .SR. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Displays the contents of the parameter stack non destructively. Very useful when debugging. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

```
0  CONSTANT  .SS
: .SL  0  '  .SS  !  ;
: .SR  -1  '  .SS  !  ;
17
```

```
: .S
```

```
CR DEPTH
```

```
IF  .SS
```

```
IF  SP@  S0  2-
```

```
ELSE  SP@  S0  SWAP  THEN
```

```
DO  I  @  0  D.  2  .SS  +-  +LOOP
```

```
ELSE  ."  EMPTY STACK  "  THEN  CR  ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: .S
  DEPTH ?DUP
  IF 0 DO DEPTH I - 1- PICK 7 U.R SPACE
    KEY? ?LEAVE LOOP
  ELSE ." Empty "
  THEN ;
```

F-PC:

```
: .S
  DEPTH 0 ABORT" Stack UNDERFLOW !! "
  DEPTH ?DUP MAX.S @ 1 <
  IF 4 MAX.S ! THEN
  IF DUP ." [" 1 .R ." ]" 0 SWAP 1- MAX.S @ 1- MIN
    DO I PICK 7 U.R SPACE -1 +LOOP
  ELSE ." Stack Empty. "
  THEN ;
```

Example:

```
1 2 3 4 .SR .S .SL .S
```

Print the current values on the stack, forwards and backwards. Use the way you like best.

Comment: Some users conceive the printed list of values to proceed from the top most value to the bottom of the stack. Other users seem to work the other way. Try .S with both and determine which is most meaningful to you.

/ n1 n2 --- n3 MVP-FORTH

Divide n1 by n2 and leave the quotient n3. n3 is rounded toward zero. (Fig) Leave the signed quotient of n1/n2. (79S) Same as MVP. (83S) n3 is the floor of the quotient of n1 divided by the divisor n2. An error condition results if the divisor is zero or if the quotient falls outside of the range {-32,768..32,767}. See: division, floored (F83) Not defined. (F-PC) Divide two signed single precision numbers and return the floored quotient. Note that -5 2 / returns a value of -3.

Pronounced: divide

Implementations:

MVP: 8088/86:

/LOOP

```
: / /MOD SWAP DROP ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
: / /MOD NIP ;
```

F-PC:

```
CODE /  
  POP BX    POP AX    CWD  
  MOV CX, BX    XOR CX, DX    0>=  
  IF    \ POSITIVE QUOTIENT CASE  
    IDIV BX    1PUSH  
  THEN  IDIV BX    OR DX, DX    0<>  
  IF    DEC AX  
  THEN  1PUSH    END-CODE
```

Example:

```
25  2  /  .
```

The two values are entered on the stack and the first is divided by the second leaving the quotient on the stack and dropping the remainder. The quotient is then printed: 12 .

Comment: Signed numbers are used with this operator. Note that division by 0 is not usually checked and the result will be unpredictable or perhaps in some implementations, an infinite loop.

NOTE: -5 2 / returns a value of -2 in Fig FORTH, MVP-FORTH, and 79S but -3 in 83S, F83, and F-PC. This could trip you!

/LOOP

n ---

MVP-FORTH

A DO-LOOP terminating word. The loop index is incremented by the unsigned magnitude of n. Until the resultant index exceeds the limit, execution returns to just after the corresponding DO, otherwise, the index and limit are discarded. Magnitude logic is used.

Note: The above definition is taken from the 79S *Reference Word Set*. However, to be consistent with the other loop functions, it has been implemented to increment until the resultant equals or exceeds the limit. (Fig) Not defined. (79S *Reference Word Set*) Same as MVP. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```

: /LOOP 3 ?PAIRS COMPILE </LOOP>
  HERE - , ; IMMEDIATE

```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```

TEST 10 1 DO I . 3 /LOOP ;

```

This ideogram must be used in a colon definition. In this example, to 9 from 1 print the value of the index and increment the index by 3. The values will be 1 4 7 .

Comment: Another variation for DO . . . LOOP control structures. This ideogram will avoid problems should the range cross from a positive to negative value as do some addresses or block numbers.

/MOD

n1 n2 --- n3 n4

MVP-FORTH

Divide n1 by n2 and leave the remainder n3 and quotient n4. n3 has the same sign as n1. (*Fig*) Leave the remainder and signed quotient of n1/n2. The remainder has the sign of the dividend. (*79S*) Same as MVP. (*83S*) n3 is the remainder and n4 the floor of the quotient of n1 divided by the divisor n2. n3 has the same sign as n2 or is zero. An error condition results if the divisor is zero or if the quotient falls outside of the range {32,768..32,767}. See: division, floored. (*F83*) Not defined. (*F-PC*) Divide two signed single precision numbers and return the floored quotient and modulus (remainder). Note that this function is designed to return a modulus which has the same sign as the denominator (usually positive), independent of the sign of the denominator.

Pronounced: divide-mod

Implementations:

MVP: 8088/86:

```

: /MOD >R S->D R> M/ ;

```

Fig: 8088/86:

:Same as MVP.

F83: 8088/86:

```
: /MOD >R S>D R> M/MOD ;
```

F -PC:

```
CODE /MOD
  POP BX    POP AX    CWD
  MOV CX, BX    XOR CX, DX    0>=
  IF    IDIV BX    2PUSH
  THEN  IDIV BX    OR DX, DX    0<>
  IF    ADD DX, BX    DEC AX
  THEN  2PUSH    END-CODE
```

Example:

```
25 3 /MOD . .
```

Enter the two values on the stack and divide the first by the second leaving the quotient on top. Then print the quotient followed by the remainder.

Comment: A convenient arithmetic operator which allows full precision and rounding in integer arithmetic. Note: Signed numbers are used and division by zero is not usually trapped, yielding unpredictable results or perhaps an infinite loop.

NOTE: -5 2 /MOD returns a value of -2 with a remainder of -1 in fig-Forth, MVP-FORTH, and 79S, but -3 with a remainder of 1 in 83S, F83, and F-PC. This could trip you!

The value is defined as an ideogram. (*Fig*) These small numbers are used so often that it is attractive to define them by name in the dictionary as constants. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Frequently used constants are faster and more code efficient. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
0 CONSTANT 0
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Same as MVP.

F-PC:

Not used.

Example:

0

Places the value of 0 on the stack. However, since it is an ideogram the value is taken from the name of the constant and not converted to the value according the the present value of BASE.

Comment: By defining common values as ideograms in the Forth dictionary, search time is decreased for the text interpreter. Also, a reference to a constant compiles just 2 bytes, while a literal would require twice that. Note that not all implementations of Forth take advantage of this capability.

0<

n --- flag

MVP-FORTH

True if n is less than zero (negative). (*Fig*) Leave a true flag if the number is less than zero (negative), otherwise leave a false flag. (*79S*) Same as MVP. (*83S*) flag is true if n is less than zero (negative). (*F83*) Returns true if top is negative, i.e. sign bit is on. (*F-PC*) Same as F83.

Pronounced: zero-less

Implementations:

MVP: 8088/86:

CODE 0<

AX POP AX AX, OR 01 AX, MOV ZLES1 JS AX DEC
HERE LABEL ZLES1 APUSH JMP END-CODE

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

CODE 0<

AX POP AX AX OR YES JS NO #) JMP END-CODE

F-PC:

CODE 0<

POP AX CWD PUSH DX NEXT END-CODE

Example:

45 0<

Place the value 45 on the stack and after the operation leave a 0 flag on the stack because the test fails. The value, 45, is lost.

Comment: One of several logical operators. Logical operators destroy

the values being tested.

```
0=      n --- flag      MVP-FORTH
```

True if n is zero. (Fig) Leave a true flag if the number is equal to zero, otherwise leave a false flag. **(79S)** Same as MVP. **(83S)** flag is true if n is zero. **(F83)** Returns True if top is zero, False otherwise. **(F-PC)** Same as F83.

Pronounced: zero-equals

Implementations:

MVP: 8088/86:

```
: 0= NOT ;
```

Fig: 8088/86:

CODE 0=

```
AX POP    AX AX, OR    1 AX MOV    ZEQU1 JZ    AX DEC
HERE LABEL ZEQU1      APUSH JMP
```

F83: 8088/86:

CODE 0=

AX POP AX AX OR YES JE NO #) JMP END-CODE

F-PC:

CODE 0=

```
POP AX      SUB AX, # 1      SBB AX, AX      1PUSH      END-CODE
```

Example:

45 0=

Place the value 45 on the stack and test it for being equal to 0. Since it is not, a 0 flag is left on the stack. The value, 45, is lost.

Comment: One of several logical operators. Logical operators destroy the values being tested.

```
0>      n --- flag                                MVP-FORTH
```

True if n is greater than zero. (*Fig*) Not defined. (*79S*) Same as MVP. (*83S*) flag is true if n is greater than zero. (*F83*) Returns true if top is positive. (*F-PC*) Same as F83.

Pronounced: zero-greater

Implementations:

MVP: 8088/86:


```
: 0> 0 > ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE 0>

```
AX POP AX AX OR YES JG NO #) JMP END-CODE
```

F-PC:

CODE 0>

```
POP AX NEG AX OV<>
```

```
IF CWD PUSH DX NEXT
```

```
THEN SHL AX, #1 1 PUSH END-CODE
```

Example:

```
45 0
```

Place the value 45 on the stack and test it for being greater than 0. Since it is, leave the flag of value 1 on the stack. The value, 45, is lost.

Comment: One of several logical operators. Logical operators destroy the values being tested.

OBRANCH

f ---

MVP-FORTH

The run-time procedure to conditionally branch. If *f* is false (zero), the following in-line parameter is added to the interpretive pointer to branch ahead or back. Compiled by IF, UNTIL, and WHILE. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

CODE OBRANCH

```
AX POP AX AX, OR ZBRAN1 JZ SI INC
```

```
SI INC NEXT JMP
```

```
HERE LABEL ZBRAN1 BRAN1 JMP END-CODE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: IF    COMPILER  OBRANCH  HERE  0  ,  2  ;  
IMMEDIATE
```

This example comes from the MVP-FORTH source code.

Comment: The compilation address of OBRANCH functions as a conditional branching opcode for the address interpreter. An in-line branch displacement must follow any compiled instance of this ideogram. These displacements are automatically generated by the IF ... ELSE ... THEN, BEGIN ... WHILE ... REPEAT and BEGIN ... UNTIL constructs. Additional user defined constructs such as a CASE may be implemented by using OBRANCH and BRANCH within new immediate compiling ideograms.

CAUTION: Executing OBRANCH directly from the terminal or screen will crash your system.

1

--- 1

MVP-FORTH

A common integer defined as a constant. (*Fig*) These small numbers are used so often that it is attractive to define them by name in the dictionary as constants. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Frequently used constants are faster and more code efficient. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
1  CONSTANT  1
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Same as MVP.

F-PC:

Not used.

Example:

```
1
```

Causes the value of 1 to be placed on the stack without having to perform the number conversion.

Comment: By defining this value as a Forth ideogram, dictionary search time and memory space are saved.

Increment n by one, according to the operation for +. (Fig) Increment n1 by 1. (79S) Same as MVP. (83S) n is the result of adding one to n according to the operation of +. (F83) Increment the top of the stack by one. (F-PC) Same as F83.

Pronounced: one-plus

Implementations:

MVP: 8088/86:

CODE 1+
AX POP AX INC APUSH JMP END-CODE

Fig: 8088/86:

$$: 1 + 1 + ;$$

F83: 8088/86:

```
CODE 1+
    AX POP    AX INC    1PUSH
```

F-PC:

Same as F83.

Example:

45 1+

Enter the value 45 on the stack and then increment it by 1 leaving the value of 46 on the stack.

Comment: A required word in 79S, which can be defined in high level Forth for portability, or in code for maximum speed.

1- n --- n-1

Decrement n by one, according to the operation -. (Fig) Not defined. (79S) Same as MVP. (83S) n is the result of subtracting one from n according to the operation of -. (F83) Decrement the top of the stack by one. (F-PC) Same as F83.

Pronounced: one-minus

Implementations:

MVP: 8088/86:

CODE 1-
AX POP AX DEC APUSH JMP END-CODE

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE 1-

AX POP AX DEC 1PUSH

F-PC:

Same as F83.

Example:

45 1-

Enter the value 45 on to the stack and then decrement the value by one.

Comment: A required word in 79S, which can be defined in high level Forth for portability, or in code for maximum speed.

2

--- 2

MVP-FORTH

A common integer value defined as a constant. (*Fig*) These small numbers are used so often that it is attractive to define them by name in the dictionary as constants. (79S) Not defined. (83S) Not defined. (F83) Frequently used constants are faster and more code efficient. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

2 CONSTANT 2

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Same as MVP

F-PC:

Not used.

Example:

2

Causes the value of 2 to be placed on the stack without using the number conversion routines. Since the ideogram is actually in the dictionary, the value can be placed on the stack even when in >BINARY.

Comment: The value is used often enough that some gain is made at interpret time by not having to search the dictionary and then CONVERT.

2! d addr ---

MVP-FORTH SUPPLEMENTAL

Store d in 4 consecutive bytes beginning at addr, as for a double number. (Fig) Not defined. (79S Extension Word Set) Same as MVP. (83S Double Number Extension Word Set) 32b is stored at addr. See: "number" (F83) Store a 32-bit value at addr. (F-PC) Same as F83.

Pronounced: two-store

Implementations:

MVP: 8088/86:

: 2! D! ;

Fig: 8088/86:

CODE 2!

```

BX POP      AX POP      AX [BX], MOV
AX POP      AX 2[BX], MOV      NEXT JMP

```

F83: 8088/86:

CODE 2!

BX POP 0 [BX] POP 2 [BX] POP NEXT

F-PC:

Same as F83.

Example: CAUTION — May corrupt or crash your system.

HEX 33.33 4AC2 2! DECIMAL

Enter the double precision value on the stack taking four bytes. Then remove these four bytes from the stack and store them in four bytes beginning at memory address 4AC2. The actual byte order within each cell is implementation dependent.

Comment: This ideogram is included in the extended double number word set of 79S. However, the ideogram D! is a better mnemonic because it avoids conflict in interpretation with the quantity 2. Thus D! is used in MVP-FORTH and 2! may be added as an alias.

2* n1 --- n2

MVP-FORTH

Leave $2 * (n1)$. (*Fig*) Not defined. (*79S Reference Word Set*) Same as MVP. (*83S*) Not defined. (*F83*) Double the number on the Stack. (*F-PC*) Same as F83.

Pronounced: two-times

Implementations:

MVP: 8088/86:

CODE 2*

AX POP AX AX, ADD APUSH JMP END-CODE

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE 2*

AX POP AX SHL 1PUSH

F-PC:

Same as F83.

Example:

45 2*

Enter the value of 45 on the stack and then double it.

Comment: A useful operation whose definition is provided in the reference vocabulary of 79S. Note that the carry and overflow are ignored.

Increment *n* by two, according to the operation for +. (*Fig*) Leave *n*1 incremented by 2. (79S) Same as MVP. (83S) *n* is the result of adding two to *n* according to the operation of +. (F83) Increment the top of the stack by two. (F-PC) Same as F83.

Pronounced: two-plus*Implementations:*

MVP: 8088/86:

CODE 2+

AX POP AX INC AX INC APUSH JMP END-CODE

Fig: 8088/86:

: 2+ 2 + ;

F83: 8088/86:

CODE 2+

AX POP AX INC AX INC 1PUSH

F-PC:

CODE 2+

POP AX ADD AX, # 2 1PUSH END-CODE

Example:

45 2+

Enter the value of 45 on the stack and then increment it by 2.

Comment: A required word in 79S, which can be defined in high level Forth for portability, or in code for maximum speed.

2- n --- n-2 MVP-FORTH

Decrement n by two, according to the operation for - . (*Fig*) Not defined. (79S) Same as MVP. (83S) n is the result of subtracting two from n according to the operation of -. (F83) Decrement the top of the stack by two. (F-PC) Same as F83.

Pronounced: two-minus

Implementations:

MVP: 8088/86:

```
CODE 2-
    AX POP    AX DEC    AX DEC    APUSH JMP    END-CODE
```

Fig: 8088/86

Not used.

F83: 8088/86

```
CODE 2-
    AX POP    AX DEC    AX DEC    1PUSH
```

F-PC:

```
CODE 2-
    POP AX    SUB AX, # 2    1PUSH    END-CODE
```

Example:

45 2-

Enter the value 45 on the stack and then decrement it by 2 .

Comment: A required word in 79S, which can be defined in high level Forth for portability, or in code for maximum speed.

2/ n1 --- n2 MVP-FORTH SUPPLEMENTAL

Leave (n1)/2. (*Fig*) Not defined. (79S *Reference Word Set*) Same as MVP. (83S) n2 is the result of arithmetically shifting n1 right one bit. The sign is included in the shift and remains unchanged. (F83) Shift the number on the stack right one bit. Equivalent to division by 2 for

positive numbers. (*F-PC*) Same as F83.

Pronounced: two-divide

Implementations:

MVP: 8088/86:

: 2/ 2 / ;

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE 2/

AX POP AX SAR 1PUSH

F-PC:

CODE 2/

POP AX SAR AX, # 1 1PUSH END-CODE

Example:

45 2/

Enter the value of 45 on the stack, halve it and drop the remainder.

Comment: A useful operation whose definition is provided in the reference vocabulary of 79S. The result is always rounded toward zero.

Leave on the stack the contents of the four consecutive bytes beginning at *addr*, as for a double number. (*Fig*) Not defined. (*79S Extension Word Sets*) Same as MVP. (*83S Double Number Extension Word Set*) 32b is the value at *addr*. See: "number". (*F83*) Fetch a 32-bit value from *addr*. (*F-PC*) Same as F83.

Pronounced: two-fetch

Implementations:

MVP: 8088/86:

: 2@ D@ ;

Fig: 8088/86:

CODE 2@

BX POP AX [BX], MOV 2 [BX] DX, MOV DPUSH JMP

F83: 8088/86:

CODE 2@

BX POP 2 [BX] PUSH 0 [BX] PUSH NEXT

F-PC:

Same as F83.

Example:

4AC2 2@

This would retrieve the double precision value, 3333, which we put at this address in the example under 2!.

Comment: This ideogram is included in the extended double number word set of 79S. However, the ideogram D@ is a better mnemonic because it avoids conflict in interpretation with the quantity 2. Thus, D@ is used in MVP-FORTH and 2@ may be added as an alias.

2CONSTANT

d --- MVP-FORTH SUPPLEMENTAL

A defining word used to create a dictionary entry for <name>, leaving d in its parameter field. When <name> is later executed, d will be left on the stack. (*Fig*) Not defined. (*79S Extension Word Sets*) Same as MVP. (*83S Double Number Extension Word Set*) A defining word executed in the form: 32b 2CONSTANT <name>. Creates a dictionary entry for <name> so that when <name> is later executed, 32b will be left on the stack. (*F83*) Create a double number constant. This is defined for completeness, but never used, so the code field is discarded. (*F-PC*) Create a double number constant.

Pronounced: two-constant

Form: d 2CONSTANT <name>

Implementations:

MVP: 8088/86:

: 2CONSTANT DCONSTANT ;

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: 2CONSTANT
  CREATE , , ( d# -- )
  DOES> 2@ ; ( -- d# ) DROP
```

F-PC:

Same as F83.

Example:

33.33 2CONSTANT NEW-VALUE

2DROP

Enter the value 33.33 which will be a double precision number and store it in an ideogram named NEW-VALUE. NEW-VALUE will then cause the 33.33 to be placed on the stack. *NOTE:* The decimal point location is not preserved in a double precision integer.

Comment: This ideogram is included in the extended double number word set of 79S. However, the ideogram DCONSTANT is a better mnemonic because it avoids conflict in interpretation with the quantity 2. Thus, DCONSTANT is used in MVP-FORTH and 2CONSTANT may be added as an alias.

2DROP

d ---

MVP-FORTH SUPPLEMENTAL

Drop the top double number on the stack. (*Fig*) Not defined. (*79S Extension Word Sets*) Same as MVP. (*83S Double Number Extension Word Set*) 32b is removed from the stack. (*F83*) Drop the top two elements of the stack. (*F-PC*) Same as F83.

Pronounced: two-drop

Implementations:

MVP: 8088/86:

```
: 2DROP  DDROP  ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
CODE 2DROP
  AX POP    AX POP    NEXT
```

F-PC:

```
CODE 2DROP
  ADD SP, # 4
  NEXT    END-CODE
```

Example:

```
2DROP
```

This ideogram will cause the top four bytes on the stack to be removed which would drop a double precision number or any other pair of single quantities.

Comment: This ideogram is included in the extended double number word set of 79S. However, the ideogram DDROP is a better mnemonic because it avoids conflict in interpretation with the quantity 2. Thus, DDROP is used in MVP-FORTH and 2DROP may be added as an alias.

2DUP

d --- d d

MVP-FORTH SUPPLEMENTAL

Duplicate the top double number on the stack. (*Fig*) Not defined. (*79S Extension Word Sets*) Same as MVP. (*83S Double Number Extension Word Set*) Duplicate 32b. (*F83*) Duplicate the top two elements of the stack. (*F-PC*) Same as F83.

Pronounced: two-dup

Implementations:

MVP: 8088/86:

: 2DUP DDUP ;

Fig: 8088/86:

CODE 2DUP

AX POP DX POP DX PUSH AX PUSH DPUSH JMP

F83: 8088/86:

CODE 2DUP

AX POP DX POP DX PUSH AX PUSH 2PUSH

F-PC:

CODE 2DUP

MOV DI, SP PUSH 2 [DI] PUSH 0 [DI]

NEXT END-CODE

Example:

33.33 2DUP

Place the double precision number on the stack taking 4 bytes and make a copy of it using the next four bytes. *NOTE:* The decimal point location is not maintained in the double precision integer.

Comment: This ideogram is included in the extended double number word set of 79S. However, the ideogram DDUP is a better mnemonic because it avoids conflict in interpretation with the quantity 2. I found one dialect which made two copies of the top of the stack! Thus, DDUP is used in MVP-FORTH and 2DUP may be added as an alias.

2OVER

MVP-FORTH SUPPLEMENTAL

d1 d2 --- d1 d2 d1

Leave a copy of the second double number on the stack. (*Fig*) Not defined. (*79S Extension Word Sets*) Same as MVP. (*83S Double Number Extension Word Set*) 32b3 is a copy of 32b1. (*F83*) Copy the second pair of numbers over the top pair. Behaves like 2SWAP for 32-bit integers. (*F-PC*) Same as F83.

2ROT

Pronounced: two-over

Implementations:

MVP: 8088/86:

: 2OVER DOVER ;

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE 2OVER

CX POP BX POP AX POP DX POP

DX PUSH AX PUSH BX PUSH CX PUSH 2PUSH

F-PC:

CODE 2OVER

MOV DI, SP PUSH 6 [DI]

PUSH 4 [DI] NEXT END-CODE

Example:

33.33 44.44 2OVER

Place the two double precision numbers on the stack and then add on a copy of the first one. NOTE: The decimal point location is not maintained in double precision integers.

Comment: This ideogram is included in the extended double number word set of 79S. However, the ideogram DOVER is a better mnemonic because it avoids conflict in interpretation with the quantity 2. Thus, DOVER is used in MVP-FORTH and 2OVER may be added as an alias.

2ROT

NOT USED IN MVP-FORTH

a b c d e f --- c d e f a b

(Fig) Not defined. (79S) Not defined. (83S *Double number Extension Word Set*) The top three double numbers on the stack are rotated, bringing the third double number to the top of the stack. (F83) Rotates top three double numbers. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: 2ROT    5 ROLL  5 ROLL  ;
```

F-PC:

Same as F83.

Example:

```
1. 2. 3.  2ROT  D. D. D.
```

Enter 3 double precision numbers, execute 2ROT and print the double numbers from the stack.

Comment: The ideogram is might better be named DROT to avoid confusion with a possible interpretation of doing the ROT function twice. Neither function is used in MVP-FORTH.

2SWAP

MVP-FORTH SUPPLEMENTAL

```
d1 d2 --- d2 d1
```

Exchange the top two double numbers on the stack. (*Fig*) Not defined. (*79S Extension word set*) Same as MVP. (*83S Double Number Extension Word Set*) The top two double numbers are exchanged. (*F83*) Swap the top two pairs of numbers on the stack. You can use this operator to swap two 32-bit integers and preserve their meaning as double numbers. (*F-PC*) Same as F83.

Pronounced: two-swap

Implementations:

MVP: 8088/86:

```
: 2SWAP  DSWAP  ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
CODE 2SWAP
```

```
    CX POP    BX POP    AX POP    DX POP .  
    BX PUSH   CX PUSH   2PUSH
```

F-PC:

Same as F83.

Example:

```
33.33  44.44  2SWAP
```

Place two double precision numbers on the stack and then exchange their positions. The decimal point location is not maintained in the double precision integer.

2VARIABLE

Comment: This ideogram is included in the extended double number word set of 79S. However, the ideogram DSWAP is a better mnemonic because it avoids conflict in interpretation with the quantity 2. Thus, DSWAP is used in MVP- FORTH and 2SWAP may be added as an alias.

2VARIABLE

MVP-FORTH SUPPLEMENTAL

A defining word used to create a dictionary entry of <name> and assign 4 bytes for storage in the parameter field. When <name> is later executed, it will leave the address of the first byte of its parameter field on the stack. (*Fig*) Not defined. (*79S Extension word set*) Same as MVP. (*83S Double Number Extension Word Set*) A defining word executed in the form: 2VARIABLE <name>. A dictionary entry for <name> is created and four bytes are ALLOTTed in its parameter field. This parameter field is to be used for contents of the variable. The application is responsible for initializing the contents of the variable which it creates. When <name> is later executed, the address of its parameter field is placed on the stack. See: VARIABLE (*F83*) Create a double length variable. This is defined for completeness, but never used, so the code field is discarded as appropriate.. (*F-PC*) Create a double length variable.

Form: 2VARIABLE <name>

Implementations:

MVP: 8088/86:

: 2VARIABLE DARIABLE ;

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: 2VARIABLE
  0 0 2CONSTANT      ( --- )
  DOES> ;            ( --- adr )  DROP
```

F-PC:

Same as F83.

Example:

```
2VARIABLE NEW-VARIABLE
```

Make a new ideogram referring to a double precision variable. Its value is not initialized.

Comment: This ideogram is included in the extended double number

word set of 79S. However, the ideogram **DVARIABLE** is a better mnemonic because it avoids conflict in interpretation with the quantity 2. Thus, **DVARIABLE** is used in MVP-FORTH and **2VARIABLE** may be added as an alias.

79-STANDARD

MVP-FORTH

Execute assuring that a 79S system is available, otherwise an error condition exists. (*Fig*) Not defined. (*79S*) Same as MVP. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

: 79-STANDARD ;

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

79-STANDARD

This ideogram does nothing in this implementation, but also does not create an error - the word is in the vocabulary.

Comment: A required ideogram in the standard, but it is unspecified how the error condition is to be generated.

:

MVP-FORTH

A defining word which selects the **CONTEXT** vocabulary to be identical to **CURRENT**. Create a dictionary entry for <name> in **CURRENT**, and set compile mode. Words thus defined are called 'colon-definitions'. The compilation addresses of subsequent words from the input stream which are not immediate words are stored into the dictionary to be executed when <name> is later executed. **IMMEDIATE** words are executed as encountered. If a word is not found after a search of the **CONTEXT** and **FORTH** vocabularies, conversion compilation of a literal number is attempted, with regard to the current **BASE**; that failing, an error condition exists. (*Fig*) Create a dictionary defining **cccc** as equivalent to the following sequence of Forth word definitions '...' until

:

the next ';' or ';CODE'. The compiling process is done by the text interpreter as long as STATE is non-zero. Other details are that the CONTEXT vocabulary is set to the CURRENT vocabulary and that words with the precedence bit set (P) are executed rather than being compiled. (79S) Same as MVP. (83S) Create a word definition for <name> in the compilation vocabulary and set the compilation state. The search order is changed so that the first vocabulary in the search order is replaced by the compilation vocabulary. The compilation vocabulary is unchanged. The test from the input stream is subsequently compiled. <name> is called a "colon definition". The newly created word definition for <name> cannot be found in the dictionary until the corresponding ; or ;CODE is successfully processed. An error condition exists if a word is not found and cannot be converted to a number or if, during compilation from mass storage, the input stream is exhausted before encountering ; or :CODE . sys is balanced with its corresponding ; . See: "compilation" "9.4 Compilation" (F83) Defines a colon definition. The definition is hidden until it is completed, or the user desires recursion. The run time for : adds a nesting level. (F-PC) Same as F83.

Form: : <name> ... ;

Pronounced: colon

Implementations:

MVP: 8088/86:

```
: :
  SP@ CSP ! CURRENT @ CONTEXT !
  CREATE SMUDGE ] <;CODE>
  DX INC BP DEC SI [BP], MOV
  DX SI, MOV NEXT JMP END-CODE
```

Fig: 8088/86:

```
: :
  ?EXECUTE !CSP CURRENT @ CONTEXT !
  CREATE [ (;CODE) DX INC BP DEC BP DEC
  SI [BP], MOV DX SI, MOV NEXT JMP
```

F83: 8088/86:

```
: :
  CREATE HIDE !CSP CURRENT @ CONTEXT !
  ] ;USES NEST ,
```

F-PC:

```
: (:)
  !CSP CURRENT @ CONTEXT !
```

```

HEADER ,JMP    XHERE PARAGRAPH +
DUP XDPSEG !   XCEG @ - ,    XDP OFF    HIDE
:USES  NEXT , -X
: :      (: ) ] ;

```

Example:

```
: TEST ;
```

Creates a new ideogram in the dictionary which in this case does nothing.

Comment: This is one of the most used ideograms in Forth. The implementations of this ideogram vary according to which vocabularies are searched and in what order. The MVP-FORTH definition is taken from 79S which implies that after the CONTEXT is searched, even if it is a daughter vocabulary, the search skips immediately to Forth.

;

MVP-FORTH

Terminate a colon-definition and stop compilation. If compiling from mass storage and the input stream is exhausted before encountering ; an error condition exists. (*Fig*) Terminate a colon-definition and stop further compilation. Compiles the run-time ;S. (79S) Same as MVP. (83S) Stops compilation of a colon definition, allows the <name> of this colon definition to be found in the dictionary, sets interpret state and compiles EXIT (or a system dependent word which performs and equivalent function). sys is balanced with its corresponding ; . See: EXIT : "stack, return" "9.4 Compilation" (F83) Terminates a colon definition. Compiles the runtime code to remove a nesting level, and changes STATE so that compilation will terminate. (F-PC) Same as F83.

Pronounced: semi-colon

Implementations:

MVP: 8088/86:

```

: ;
  ?CSP COMPILE EXIT SMUDGE
  [COMPILE] [ ; IMMEDIATE

```

Fig: 8088/86:

```

: ;
  ?CSP COMPILE ;S SMUDGE [COMPILE] [ ;
  IMMEDIATE

```

F83: 8088/86:

;CODE

```
: ;  
  ?CSP   COMPILE UNNEST   REVEAL   [compile] [ ;  
  IMMEDIATE
```

F-PC:

```
: ;  
  STATE @ 0= ABORT" Not Compiling!"  
  ?CSP   COMPILE UNNEST   REVEAL   [COMPILE] [ ;  
  IMMEDIATE
```

Example:

```
: TEST ;
```

Used to end the colon definition which in this case does nothing.

Comment: In this implementation, the 79S error condition aborts with an "INPUT STREAM EXHAUSTED" message.

;CODE

NOT USED IN MVP-FORTH

Used in the form:

```
: <name> ... ;CODE.
```

Stop compilation and terminate a defining word <name> ASSEMBLER becomes the CONTEXT vocabulary. When <name> is executed in the form:

```
<name> <namex>
```

to define the new <namex>, the execution address of <namex> will contain the address of the code sequence following the ;CODE in <name>. Execution of any <name> will cause this machine code sequence to be executed. (*Fig*) Stop compilation and terminate a new defining word cccc by compiling(;CODE). Set the CONTEXT vocabulary to ASSEMBLER, assembling to machine code the following mnemonics. When cccc is later executed ... the word nnnn will be created with its execution procedure given by the machine code following cccc. That is, when nnnn is executed, it does so by jumping to the code after nnnn. An existing defining word must exist in cccc prior to ;CODE. (79S Assembler Word Set) Same as MVP. (83S Assembler Extension Word Set) Used in the form: : <namex> ... <create> ... ;CODE ... END-CODE. Stops compilation, terminates the defining word <namex> and executes ASSEMBLER. When <namex> is executed in the form: <namex> <name>, to define the new <name>, the execution address of <name> will contain the address of the code sequence following the ;CODE in <namex>. Execution of any <name> will cause this machine code sequence to be executed. sys1 is balanced with its corresponding:

. sys2 is balanced with its corresponding END-CODE . See: CODE DOES> (F83) Used for defining the run time portion of a defining word in low level code. (F-PC) Same as F83.

Pronounced: semi-colon-code

Implementations:

MVP: 8088/86:

```
: ;CODE
  ?CSP COMPILE <;CODE> [COMPILE] [
    [COMPILE] ENTERCODE ; IMMEDIATE
```

Fig: 8088/86:

```
: ;CODE
  ?CSP COMPILE (;CODE) [COMPILE] [ NOOP ;
  Note: Replace the NOOP with your ASSEMBLER.
```

F83: 8088/86:

```
: ;CODE
  ?CSP COMPILE (;CODE) [COMPILE] [
  REVEAL ASSEMBLER ; IMMEDIATE
```

F-PC:

```
: ;CODE
  ?CSP COMPILE (;CODE) HERE X, [COMPILE] [
  REVEAL SETASSEM ; IMMEDIATE
```

Example:

```
: USER CONSTANT ;CODE D NX XCHG M E MOV
  0 D MVI UP LHL D DAD HPUSH JMP END-CODE
```

In this example, after the USER variable is given a name in a colon definition, we switch to the ASSEMBLER which is not included in this implementation but is referenced by the CROSS-COMPILER. This begins the specification of the run-time activity for USER variables.

Comment: This ideogram functions like DOES>, except that the generated offspring's code address is redirected to usable machine code rather than to a call to the subroutine, DODOES. It should be included as part of an ASSEMBLER vocabulary.

;S ---

NOT USED IN MVP-FORTH

(Fig) Stop interpretation of a screen. ;S is also the run-time word compiled at the end of a colon-definition which returns execution to the calling procedure. (79S Reference Word Set) Stop interpretation of a

<

block. For execution only. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

CODE ;S [BP] SI, MOV BP INC BP INC NEXT JMP

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

: ; COMPILE ;S [COMPILE] [; IMMEDIATE

This simplified implementation of semicolon compiles the ideogram and sets the execution mode. No error checking or unsmudging is performed.

Comment: This ideogram is now obsolete, having been replaced by EXIT.

<

n1 n2 --- flag

MVP-FORTH

True if n1 is less than n2. (*Fig*) Leave a true flag if n1 is less than n2; otherwise leave a false flag. (79S) Same as MVP. (83S) Flag is true if n1 is less than n2. (F83) Compare the top two elements on the stack as signed integers and return true if $n1 < n2$. (F-PC) Same as F83.

Pronounced: less-than

Implementations:

MVP: 8088/86:

CODE <

DX POP AX POP DX BX, MOV AX BX, XOR
LES1 JS DX AX, SUB
HERE LABEL LES1 AX AX, OR # 0 AX, MOV
LES2 JNS AX INC
HERE LABEL LES2 APUSH JMP END-CODE

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE <
  AX POP    BX POP    AX BX CMP    YES JL
  NO #) JMP END-CODE
```

F-PC:

```
CODE <
  POP AX    POP BX    CMP BX, AX  >=
  IF  SUB AX, AX    1PUSH  THEN
  MOV AX, # TRUE    1PUSH    END-CODE
```

Example:

```
45 47 <
```

Enter two integers on the stack and then test to see if the first is less than the second. In this example it is and the value of the flag on the stack is set to 1. The test destroys both operands.

Comment: The limits for this signed comparison are strictly defined in 79S: -32768 32767 < must return true and -32768 and 0 < must be distinguished.

<# d1 --- d1 MVP-FORTH

Initialize pictured numeric output. The ideograms (words) <#, #, #S, HOLD, SIGN, and #> can be used to specify the conversion of a double-precision number into an ASCII character string stored in right-to-left order. (*Fig*) Setup for pictured numeric output formatting using the words: <# # #S SIGN #>. The conversion is done on a double number producing text at PAD. (*79S*) Same as MVP. (*83S*) Initialize pictured numeric output conversion. The words: <# # #S #> HOLD SIGN can be used to specify the conversion of a double number into an ASCII text string stored in right-to-left order. (*F83*) Start numeric conversion. (*F-PC*) Same as F83.

Pronounced: less-sharp

Implementations:

MVP: 8088/86:

```
: <#    PAD  HLD  !  ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Same as MVP.

F-PC:

<-FIND>

Same as MVP.

Example:

```
45.  <#  #S  #>  TYPE
```

Enter the double precision value 45. on the stack, then format the value for printing it and finally print it.

Comment: In 79S there is no specification as to where the pictured number will be stored. Also, implementations vary on how the sign of negative numbers is handled.

<+LOOP>

n ---

MVP-FORTH

The run-time procedure compiled by +LOOP, which increments the loop index by n and tests for loop completion. See +LOOP. (*Fig*) Not defined. (79S) Not defined. (83S) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
CODE <+LOOP>
  BX POP  ALOOP1 JMP  END-CODE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
  : +LOOP
  3  ?PAIRS  COMPILE  <+LOOP>  HERE  -  ,  ;  IMMEDI-
ATE
```

The example illustrates the use of this run time procedure.

Comment: Although dangerous and almost totally useless for applications, this ideogram is available to the programmer. Like BRANCH and 0BRANCH, it expects an in-line branching displacement.

--- pfa b tf (if found)

<-FIND>

MVP-FORTH

--- ff (if not found)

Accepts the next word (delimited by blanks) in the input stream to HERE and searches the CONTEXT and then the FORTH vocabularies for a matching entry. If found, the dictionary entry's parameter field address, its length byte, and a boolean true is left. Otherwise, only a boolean false is left. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: <-FIND>
  BL WORD CONTEXT @ @ <FIND> ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
<-FIND> JUNK
```

Search the dictionary for the ideogram JUNK. Presumably, the ideogram is not in the dictionary and the flag value of 0 is left on the stack.

Comment: This ideogram is vectored from -FIND via the user variable '-FIND. Note: The null character used to terminate the terminal and disk buffers is defined as an ideogram in the dictionary (see X). This may occasionally produce bewildering error messages or unexpected results when you use the ideograms: -FIND, ' , [COMPILE] , FORGET , or others which search the dictionary or define new words.

<. ">

MVP-FORTH

A run-time procedure, compiled by ." which transmits the following in-line text to the selected output device. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: <. ">
  R@ COUNT DUP 1+ >R + >R TYPE ;
```

</LOOP>

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example: CAUTION: Execution outside a colon definition will crash the system.

```
: ."
BLK @
IF BLK @ BLOCK ELSE TIB @ THEN
>IN @ + C@ 22 = NOT
IF 22 STATE @
  IF <."> WORD C@ 1+ ALLOT
  ELSE WORD COUNT TYPE THEN
ELSE 1 >IN +! THEN ;
```

This example is an alternate implementation of .".

Comment: This synonym for fig-Forth's (.), is used in MVP-FORTH in order to avoid confusion with comments within parentheses.

</LOOP>

u ---

MVP-FORTH

The run-time procedure compiled by /LOOP, which increments the loop index by u and tests for loop completion. See /LOOP . (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
CODE </LOOP>
  BX POP  BX BP, ADD  AUPLO1 JB  [BP] AX, MOV
  2[BP] AX, SUB  JNB AUPLO1  JMP BRAN1
  HERE LABEL AUPLO1  # 4H BP, ADD
  SI INC  SI INC  NEXT JMP  END-CODE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: /LOOP
  3 ?PAIRS COMPILE </LOOP> HERE - , ; IMMEDIATE
```

The example illustrates the use of this run-time procedure.

Comment: Although dangerous and almost totally useless for applications, this ideogram is available to the programmer. Like **BRANCH** and **0BRANCH**, it expects an inline branching displacement. It differs from **<+LOOP>** in that the index is unsigned which is desirable for address looping.

<;CODE>

MVP-FORTH

The run-time procedure, compiled by **<;CODE>**, that rewrites the code field of the most recently defined word to point to the following machine code sequence. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: <;CODE>
  R> LATEST PFA CFA ! ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example: CAUTION — Execution outside a colon definition will crash the system.

```
HEX
: DOES>
  ?CSP COMPILE <;CODE> E8 C,
  COMPILE [ HERE 4 + , ] ;
DECIMAL
```

This is the only use of this word. It provides the necessary function for the revised **DOES>** as used in *79S*. Note that the literal hex value **E8**, is the 8088/86 specific op-code which is compiled at this point.

<>

Comment: This ideogram is needed by DOES>, and therefore appears even in systems which lack an assembler vocabulary and the ideogram, ;CODE. The functional definition differs from that given by fig-Forth. The latter includes two extra bytes between the code field address and the parameter field address of every DOES> word. The 79S function makes the format of all compiled definitions more consistent.

<<CMOVE> addr1 addr2 u --- MVP-FORTH

The primitive code routine for <CMOVE. It can move up to 65535 bytes. Nothing is moved if u = 0. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
CODE <<CMOVE>
  SI BX, MOV    CX POP    DI POP    CX DI, ADD
  DI DEC    SI POP    CX SI, ADD    SI DEC
  STD    MOVSB REP    CLD    BX SI, MOV
  NEXT JMP    END-CODE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: <CMOVE
  DUP    1    <R
  IF DDROP DROP ELSE <<CMOVE> THEN ;
```

This example is from the MVP-FORTH source code.

Comment: This is the primitive for the function which proceeds from high memory towards low memory.

<> n1 n2 --- f NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined (F83) Returns true if the two element are not equal, else false. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: <>
  NOT ;
```

F-PC:

```
CODE <>
  POP AX POP BX CMP BX, AX 0=
  IF SUB AX, AX 1PUSH
  THEN MOV AX, # TRUE 1PUSH END-CODE
```

Example:

```
1 2 <>
```

The example returns a true flag because the values are not equal.

Comment: This simple ideogram can be included in either high level or in code if there is occasion to use it in your particular application.

<?TERMINAL>

--- f

MVP-FORTH

The run-time procedure which performs a test of the terminal keyboard for actuation of the break key. A true flag indicates acuation. This definition is installation dependent. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: <?TERMINAL>
  0600 0 0 0FF 21 INTCALL 00FF AND ;
```

Fig: 8088/86:

```
CODE ?TERMINAL
PQTER JMP
```

F83: 8088/86:

Not used.

F-PC:

Not used.

<ABORT">

Example:

```
' <?TERMINAL> CFA '?TERMINAL !
```

The example illustrates assigning the run-time code field address to a user variable for vectoring. It is taken from the source code.

Comment: This installation dependent function can not be made the same for all implementations of MVP-FORTH. The implementation for the IBM is more complicated than some. It is necessary to probe the type-ahead input buffer and modify it. Since the character struck would otherwise be lost, its ASCII value is left as the true flag. The system flag is automatically cleared. Under the 8080 CP/M implementation there is no type-ahead buffer. The flag must be cleared with KEY DROP.

<ABORT">

f ---

MVP-FORTH

The run-time procedure used with ABORT" . (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: <ABORT">
  IF WHERE CR R@ COUNT TYPE
  SP! QUIT
  ELSE R> DUP C@ + 1+ >R
  THEN ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: ABORT"
  ?COMP COMPILE <ABORT"> 'STREAM C@ 22 =
  IF 1 >IN +! 0 C,
  ELSE 22 WORD DUP C@ 1+ SWAP OVER
  + C@ 22 = NOT ?STREAM ALLOT
  THEN ; IMMEDIATE
```

This example is taken from the MVP-FORTH source code.

Comment: This ideogram takes an inline string which is set up by the immediate compiling ideogram **ABORT**". Although dangerous and almost totally useless for applications, it is available to the programmer.

<ABORT>

MVP-FORTH

Clear the data and return stacks, setting execution mode. Return control to the terminal. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

: **<ABORT>**

SP! ?STACK COMPILE FORTH DEFINITIONS QUIT ;

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

*Example:***<ABORT>**

This simple ideogram will clear the stack and return to Forth definitions so that you can begin again. Your definitions are not forgotten.

Comment: This ideogram is vectored from **ABORT** via variable ' **ABORT**. In some versions of Forth, this ideogram also includes a statement of the name of the version and other information, while in *79S*, no indication is made.

<BLOCK>

n --- addr

MVP-FORTH

Leave the address of the first byte in block *n*. If the block is not already in memory, it is transferred from mass storage into whichever memory buffer has been least recently accessed. If the block occupying that buffer has been **UPDATED** (i.e. modified), it is rewritten onto mass storage before block *n* is read into the buffer. *n* is an unsigned number. If correct mass storage read or write is not possible, an error condition exists. Only data within the latest block referenced by **<BLOCK>** is valid by byte address, due to sharing of the block buffers. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not

<BUILDS

defined.

Implementations:

MVP: 8088/86:

```
: <BLOCK>
  OFFSET @ + >R PREV @ DUP @ R@ - 2*
  IF
    BEGIN +BUF NOT
    IF DROP R@ BUFFER DUP R@ 1 R/W 2-
    THEN DUP @ R@ - 2* NOT
    UNTIL DUP PREV !
  THEN R> DROP 2+ ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
45 <BLOCK>
```

Entering the desired block number followed by the ideogram leaves the address of the beginning of that block on the top of the stack. If it is not already in a buffer, the block will be read in from disk.

Comment: This ideogram is vectored from BLOCK via user variable 'BLOCK. It is a useful ideogram which speeds access to disk information when it is not modified. MVP-FORTH has fixed a problem in some earlier implementations of fig-Forth, which sometimes failed to update the last two lines of a source screen on disk.

<BUILDS

NOT USED IN MVP-FORTH

(Fig) Used within a colon-definition: ... Each time ccc is executed, <BUILDS defines a new word with a high-level execution procedure. Executing cccc ... uses <BUILDS to create a dictionary entry for nnnn with a call to the DOES> part for nnnn. When nnnn is later executed, it has the address of its parameter area on the stack and executes the words after DOES> in cccc. <BUILDS and DOES> allow run-time preceedures to [be] written in high-level rather than in assembler code (as required by :CODE). (79S Reference Word Set) Used in conjunction with DOES> in defining words. When <name> executes,

creates a dictionary entry for the new <namex>. The sequence of words between <BUILDS and DOES> establishes a parameter field for <namex>. When <namex> is later executed, the sequence of words following DOES> will be executed, with the parameter field address of <namex> on the data stack. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Form: : <name> ... DOES> ... ;
 <name> <namex>

MVP: 8088/86: Could be defined as:

: <BUILDS CREATE ;

Fig: 8088/86:

: <BUILDS 0 CONSTANT ;

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

: CONSTANT <BUILDS , DOES> @ ;

This example illustrates a high level implementation of CONSTANT. The built-in implementation utilizing ;CODE and assembly language will run faster.

Comment: Because the word is so well established and has some beauty of construct in conjunction with DOES>, it may be defined as an alias of CREATE. The definition is modified from that in fig-Forth and older programs and may not always work.

<CMOVE

addr1 addr2 n ---

MVP-FORTH

Copy n bytes beginning at addr1 to addr2. The move proceeds within the bytes from high memory toward low memory. (Fig) Not defined. (79S Reference Word Set) Same as MVP. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Pronounced: reverse-c-move

Implementations:

MVP: 8088/86:

: <CMOVE
 DUP 1 <
 IF DDROP DROP

<CMOVE>

ELSE <<CMOVE> THEN ;

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

HEX 4AC3 4AC4 20 <CMOVE DECIMAL

With the normal CMOVE the data from 4AC4 up would be written over with the value in 4AC3 and the information would be lost. Instead, <CMOVE shifts 32 bytes one position to the right.

Comment: This ideogram works exactly like <CMOVE>, except when its source and destination fields overlap. The ideogram BMOVE, a better solution, intelligently chooses the action so that overlapping fields hold no surprises for the programmer.

<CMOVE>

addr1 addr2 u ---

MVP-FORTH

The primitive code routine for CMOVE and MOVE. Up to 65,535 bytes may be moved. Nothing is moved when u = 0. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

CODE <CMOVE>

CX POP DI POP SI POP

DS AX, MOV AX ES, MOV MOVSB REP

BX SI, MOV NEXT JMP END-CODE

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

: CMOVE


```
DUP 1 <
IF DROP DROP DROP
ELSE <CMOVE>
THEN ;
```

The example uses this procedure which remains available to the programmer.

Comment: This version of a byte move utility was developed early and was at one time used in the definition of FILL by taking advantage of its overwriting property. In MVP-FORTH, it is not used in this way. There really is little reason to use it instead of BMOVE, which moves bytes without overwriting them - an intelligent CMOVE.

<CR>

MVP-FORTH

Cause a carriage-return and line-feed to occur at the current output device, as configured by the system. The user variable OUT is reset to zero. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

HEX

: <CR> 0D EMIT 0A EMIT 0 OUT ! ;

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

<CR> <CR> ." HELLO"

The two ideograms cause the display to advance two lines and then print HELLO at the left margin of the display.

Comment: This ideogram is vectored from CR via user variable 'CR. Placing the ideogram, <CR>, in the input stream is not equivalent to pressing the "Return" key on the terminal keyboard.

<EMIT>

<DO>

n1 n2 ---

MVP-FORTH

The run-time procedure compiled by DO which moves the loop control parameters to the return stack. See DO. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

CODE <DO>

DX POP AX POP SP BP, XCHG

AX PUSH DX PUSH SP BP, XCHG

NEXT JMP END-CODE

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: DO COMPILE <DO> HERE 3 ; IMMEDIATE
```

The example uses the run-time procedure. It remains available to the programmer.

Comment: The ideogram <DO> has been used as the primitive in MVP-FORTH in place of (DO) which is used in fig-Forth. The change was made to avoid use of parentheses.

<EMIT>

c ---

MVP-FORTH

Transmit a character to the currently defined output port according to the configuration of the system. The user variable OUT is increased by one. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

HEX

: <EMIT>

EPRINT @

IF DUP 0500 0 ROT 0 SWAP 21 INTCALL DROP THEN

0E00 PLUS 07 0 0 10 INTCALL DROP

```
1 OUT +! ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
65 <EMIT>
```

Entering the decimal value 65 followed by this ideogram will cause the character A to be printed.

Comment: This ideogram is vectored from EMIT via the user variable 'EMIT.

<EXPECT>

addr n ---

MVP-FORTH

The run-time routine executed by EXPECT. It transfers characters from the terminal beginning at addr, upward, until a "return" or the count of n has been received. Take no action for n less than or equal to zero. One or two nulls are added at the end of the text. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: <EXPECT>
  OVER + OVER
  DO KEY DUP 8 = OVER 7F = OR
  IF DROP DUP I = DUP R> 2- + >R
  IF BELL
  ELSE BSOUT DUP EMIT
  20 EMIT FFFD OUT +! THEN
  ELSE DUP 0D =
  IF LEAVE DROP BL 0
  ELSE DUP
  THEN I C! 0 I 1+ !
  THEN EMIT 1
/LOOP DROP ;
```

Fig: 8088/86:

<FILL>

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
' <EXPECT> CFA 'EXPECT !
```

Store the code field address of the run-time routine <EXPECT> in the user variable 'EXPECT. It is then ready to be vectored by EXPECT.

Comment: Because EXPECT has features which may be installation dependent, it has been vectored. This ideogram is the run-time routine which includes some features, such as backspace handling.

<FILL>

addr n b ---

MVP-FORTH

A primitive for FILL which executes the actual function if selected. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
CODE <FILL>
```

```
AX POP CX POP DI POP
```

```
DS BX, MOV BX ES, MOV CLD STOSB REP
```

```
NEXT JMP END-CODE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: FILL
```

```
OVER 0
```

```
IF <FILL> ELSE DROP DROP DROP THEN ;
```

The example illustrates the use of the ideogram in taking no action if zero bytes are to be moved.

Comment: This implementation of <FILL> utilizes its own code and not the high level CMOVE.

<FIND>

MVP-FORTH

```
addr1 addr2 --- pfa b tf    (ok)
addr1 addr2 --- ff          (bad)
```

Searches the dictionary starting at the name field address addr2, matching to the text at addr1. Returns the parameter field address, length byte of name field and boolean true for a good match. If no match is found, only a boolean false is left. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
HEX
CODE <FIND>
    DS AX, MOV
    AX ES, MOV    BX POP    CX POP
    HERE LABEL AFIND1    CX DI, MOV    [BX] AL, MOV
    AL DL, MOV    [DI] AL, XOR
    3F AL, AND    AFIND3 JNZ
    HERE LABEL AFIND2    BX INC    DI INC    [BX] AL, MOV
    [DI] AL, XOR    AL AL, ADD
    AFIND3 JNZ    AFIND2 JNB
    5 BX, ADD    BX PUSH    1 AX,MOV
    DH DH, SUB    DPUSH JMP
    HERE LABEL AFIND3    BX INC    AFIND4 JB
    [BX] AL, MOV    AL AL, ADD    AFIND3 JMP
    HERE LABEL AFIND4    [BX] BX, MOV
    BX BX, OR    AFIND1 JNZ
    0 AX MOV    APUSH JMP    END-CODE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: -FIND    BL    WORD    CONTEXT    @    @    <FIND>    ;
```

<INTERPRET>

This example comes from the MVP-FORTH implementation source code.

Comment: This is the primitive used in MVP-FORTH in place of (FIND) in order to avoid confusion with comments. It provides a means of searching the dictionary without using the input stream.

<INTERPRET>

MVP-FORTH

Begin interpretation at the character indexed by the contents of >IN relative to the block number contained in BLK, continuing until the input stream is exhausted. If BLK contains zero, interpret characters from the terminal input buffer. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: <INTERPRET>
  BEGIN -FIND
    IF STATE @ <
      IF CFA , ELSE CFA EXECUTE THEN
    ELSE HERE NUMBER DPL @ 1+
      IF [COMPILE] DLITERAL
        ELSE DROP [COMPILE] LITERAL
      THEN
    THEN ?STACK
  AGAIN ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: <LOAD>
  BLK @ >R >IN @ >R 0 >IN !
  BLK ! ,INTERPRET>
  R> >IN ! R> BLK ! ;
```

This example is modified from the MVP-FORTH implementation.

Comment: This ideogram is vectored from INTERPRET via the user

variable 'INTERPRET. It is used to interpret text source >IN MVP-FORTH. The sequence "STATE @" is sneaky. It returns a true flag only if the compilation mode is set and the ideogram located by -FIND is not immediate. <INTERPRET> is then written as an infinite loop which exits implicitly at the end of an input line or disk screen.

<KEY>

--- char

MVP-FORTH

Leave the ASCII value of the next available character from the current input device, according to the configuration of the system. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

HEX

: <KEY>

BEGIN 0700 0 0 0 21 INTCAL

OFF (hex not off) AND DUP 10 - NOT

IF DROP 1 EPRINT @ XOR EPRINT ! THEN

REPEAT ;

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

<KEY>

Execution of this ideogram causes the program to wait for any single input from the keyboard and upon receiving it places the ASCII value of the input on the stack.

Comment: This ideogram is vectored from KEY via user variable 'KEY. It provides a way of finding out the ASCII value of characters without reference to a chart. It may also be used in selecting from a menu requiring only a single character input or for a wait until any character is input from the terminal. The internal details of <KEY> are installation dependent.

<LOAD>

<LINE> n1 n2 --- addr count MVP-FORTH

Convert the line number n1 and the screen n2 to the disc buffer address containing the data. A count of 64 indicates the full line text length. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: <LINE>    BLOCK   SWAP   C/L   *   +   C/L ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: .LINE    <LINE>    -TRAILING   TYPE   ;
```

This simple example comes from the MVP-FORTH source code.

Comment: This primitive is useful in a variety of manipulations among lines on various screens and to identify lines for searching with **-TEXT** and **MATCH**, for example.

<LOAD> n --- MVP-FORTH

Begin interpretation of screen n by making it the input stream; preserve the locators of the present input stream (from **>IN** and **BLK**). If interpretation is not terminated explicitly it will be terminated when the input stream is exhausted. Control then returns to the input stream containing **LOAD**, determined by the input stream locators **>IN** and **BLK**. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Load the current file starting at the current file offset.

Implementations:

MVP: 8088/86:

```
: <LOAD>
  ?DUP   NOT
  ABORT"   UNLOADABLE"
  BLK   @   >R   >IN   @   >R
```

```
0 >IN ! BLK ! INTERPRET
R> >IN ! R> BLK ! ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

See SEQREAD.SEQ File.

Example:

```
45 <LOAD>
```

This example will start loading the contents of Screen 45.

Comment: This ideogram is vectored from LOAD via user variable 'LOAD. Screens which end with several blank lines will load faster if the ideogram, EXIT, appears following the last definition or operation. Also, one can avoid loading a whole screen without erasing the undesired contents by terminating the desired source with the ideogram. This technique is not sanctioned by 79S and is implementation dependent. Note: This implementation does not permit loading block zero.

<LOOP>

MVP-FORTH

The run-time procedure compiled by LOOP which increments the loop index and tests for loop completion. See LOOP. (*Fig*) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

HEX

CODE <LOOP>

```
01 BX, MOV
HERE LABEL ALOOP1 [B] CX, MOV
BX [BP], ADD ALOOP2 JS
CX AX, MOV BX AX, AND ALOOP3 JNS
HERE LABEL LOOP2 CX AX, MOV
BX AX, OR BLOOP8 JNS
HERE LABEL ALOOP3 [BP] AX, MOV [BP] 2+ AX, SUB
ALoop7 JS [BP] AX, MOV AX AX, AND ALOOP5 JS
HERE LABEL ALOOP4 BX AX, MOV
AX AX, AND ALOOP8 JNS JMP BRAN1
```

<MARK

```
HERE LABEL ALOOP5    [BP] 2+ AX, MOV
  AX AX, AND    ALOOP4 JS
HERE LABEL ALOOP6    BX AX, MOV
  AX AX, AND    ALOOP8 JS    BRAN1 JMP
HERE LABEL ALOOP7    [BP] AX, MOV
  AX AX, AND    ALOOP6 JS    ALOOP5 JMP
HERE LABEL ALOOP8    4 BP, ADD
  SI INC    SI INC    NEXT JMP    END-CODE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: LOOP    3    ?PAIRS    COMPILE    <LOOP>    HERE    -    ,    ;
IMMEDIATE
```

This example from the MVP-FORTH source code illustrates the only use of <LOOP> which, though dangerous, remains available to the programmer.

Comment: This is a primitive similar to (LOOP) in fig-Forth but renamed to avoid confusion with comments and modified to conform with the requirements of 79S. Any compiled instance of must be followed by an in-line branching displacement. This is the main use of this primitive though it is available to the programmer.

<MARK

--- addr

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S *System Extension Word Set Glossary*) Used at the destination of a backward branch. addr is typically only used by <RESOLVE to compile a branch address. (F83) Set up for a backwards branch. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: <MARK    HERE    ;
```

F-PC:

```
: <MARK    XHERE NIP ;
```

Example:

```
: ?<MARK    TRUE    <MARK    ;
```

The example sets up a backwards branch with error checking in F-PC.

Comment: The ideogram from the 83 Standard System Extension Word Set Glossary provides one way to handle backward branching.

<NUMBER>

addr --- d

MVP-FORTH

Convert the count and character string at addr , to a signed 32-bit integer, using the current base. If numeric conversion is not possible, an error condition exists. The string may contain a preceding negative sign. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: <NUMBER>
  0 0 ROT DUP 1+ C@
  AMINUS = DUP >R + -1 DPL !
  CONVERT DUP C@ BL >
  IF DUP C@ ADOT = NOT
    ABORT" NOT RECOGNIZED" 0 DPL !
    CONVERT DUP C@ BL >
    ABORT" NOT RECOGNIZED"
  THEN DROP R>
  IF DNEGATE THEN ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

(AMINUS is the ASCII value of " - ")

(ADOT is the ASCII value of " . ")

(These are compiled as literals by the cross-compiler.)

<PAGE>

Example:

```
: INPUT
  ." INPUT AN INTEGER --- "
  QUERY BL WORD <NUMBER> DROP ;
```

This definition provides for a prompt and then a pause for the operator to input the requested integer. Then the input character stream is parsed, converted to a double precision value and reduced to a single precision value left on the stack.

Comment: The version and implementation of this ideogram in MVP-FORTH conforms with that in the 79S reference word set and fig-Forth. It will recognize two non-numeric characters: a decimal point and a leading negative sign. The position of the decimal point is recorded in the user variable DPL. This feature enables a user program to scale or adjust the converted value as desired. It will give an error message if any other special character is used. Note that the definition given in *Starting Forth* is different.

<PAGE>

MVP-FORTH

Clear the terminal screen or perform an action suitable to the output device currently active. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
HEX
: <PAGE> 0D EMIT 0A EMIT ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

(Note: In the distribution version of MVP-FORTH, a fail-safe implementation produces a carriage return rather than the expected function. Since this is a vectored function, the user can add his own CLEAR SCREEN sequence.)

Example:

```
: <IBM> 3 0 0 0 10 INTCALL DROP ;
```

' <IBM> CFA 'PAGE ! PAGE

Entering this ideogram will now clear the terminal screen on an IMB.

Comment: This ideogram is vectored from PAGE via user variable 'PAGE. It allows one to start with a clear screen and the cursor at home. Note that in spite of a common operating system such as DOS, not all terminals will use this code to clear the screen. This ideogram can be redefined in high level Forth for the particular terminal and the new code field address.

<R/W> addr blk f --- MVP-FORTH DISK I-O

The fig-Forth standard disk read-write linkage. addr specifies the source or destination buffer address, (not necessarily the Forth buffer), blk is the sequential number of the referenced block; and f is a flag for f = 0 write and f = 1 read. <R/W> determines the location on mass storage, performs the read-write and performs any error checking. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: <R/W>
  USE @ >R ROT USE ! SWAP MAX-DRV 0
  DO I DR-DEN DENSITY !
  DUP BPDRV - -1 >
  IF BPDRV - I 1+ MAX-DRV =
    IF R> R> DDROP R> USE !
    1 ABORT" BLOCK OUT OF RANGE" THEN
  ELSE I DRIVE ! LEAVE
  THEN
  LOOP SPBLK * SPBLK 0
  DO DDUP T&SCALC
  IF SEC-READ
  ELSE SEC-WRITE
  THEN 1+ HDBT 4 - SPBLK / USE +1
  LOOP
  DDROP R> USE ! ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

<RESOLVE

F-PC:

Not used.

Note: HDBT is a cross compiler equate for 1028, the block size plus its header and trailer.

Example:

```
: BUFFER
  USE @ DUP >R
  BEGIN +BUF UNTIL
  USE ! R@ @ 0
  IF R@ 2+ R@ @ 7FFF AND 0 <R/W> THEN
  R@ ! R@ PREV ! R> 2+ ;
```

This definition is modified from the MVP-FORTH source code. It is one of the principal uses of the ideogram, <R/W>.

Comment: This ideogram is a primitive in many implementations of Forth. If it is available, it is possible to read and write from disk to any area in memory such as a special buffer, without going through the regular block buffers. Of course such a procedure is installation dependent and prohibited by 79S.

<RESOLVE

addr ---

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S *System Extension Word Set Glossary*) Used at the source of a backward branch after either BRANCH or ?BRANCH. Compiles a branch address using addr as the destination address. (F83) Resolve a Backwards Branch. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: <RESOLVE , ;
```

F-PC:

```
: <RESOLVE X, ;
```

Example:

```
: ?<RESOLVE SWAP ?CONDITION <RESOLVE ;
```

The example resolves a backwards branch with error checking in F-PC.

Comment: The ideogram from the 83S *System Extension Word Set Glossary* provides one way to handle backward branching.

<T&SCALC>

n ---

MVP-FORTH DISK I-O

Track, sector and drive calculation for disk I-O. n is the total sector displacement. The corresponding track, and sector numbers are calculated. The track number is stored in TRACK; the sector number is stored in SEC. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: <T&SCALC>
  DENSITY @ 1 =
  IF SPT 2* /MOD SWAP
    SPT /MOD
    IF 140 +
      THEN SWAP SPT * +
    THEN SEC ! ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Note: Most DOS systems start counting sectors at 1. However, a few start at sector 0. It is a simple patch to add a no-op, NOOP, to Forth and replace the code field address of 1+ in T&SCALC with the code field address of NOOP.

Example:

```
: <R/W>
  USE @ >R ROT USE ! SWAP MAX-DRV 0
  DO I DR-DEN DENSITY !
  DUP BPDRV - -1 >
  IF BPDRV - I 1+ MAX-DRV =
    IF R> R> DDROP R> USE !
    1 ABORT" BLOCK OUT OF RANGE" THEN
  ELSE I DRIVE ! LEAVE
  THEN
```

<VOCABULARY79>

```
LOOP   SPBLK * SPBLK 0
DO     DDUP T&SCALC
      IF SEC-READ
      ELSE SEC-WRITE
      THEN 1+ HDBT 4 - SPBLK / USE +!
LOOP
DDROP  R> USE ! ;
```

This example is modified from the MVP-FORTH implementation.

Comment: This is a revised implementation from that in fig-Forth; it takes into account the number of sectors which are present on each disk in making the calculation. The DOS implementation requires only the sector value.

<VOCABULARY79>

MVP-FORTH

The run-time routine for a defining word to create (in the CURRENT vocabulary) a dictionary entry for <name>, which specifies a new ordered list of word definitions. Subsequent execution of <name> will make it the CONTEXT vocabulary. When <name> becomes the CURRENT vocabulary (See DEFINITIONS), new definitions will be created in that list. In lieu of any further specifications, new vocabularies chain to Forth. That is, when a dictionary search through a vocabulary is exhausted, Forth will be searched. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Form: VOCABULARY <name>

Implementations:

MVP: 8088/86:

```
: <VOCABULARY79>
  CREATE A081 , ' FORTH ,
  HERE VOC-LINK @ , VOC-LINK !
DOES> 2+ CONTEXT ! ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
' <VOCABULARY79> CFA 'VOCABULARY !
```

The example places the code field address of the run-time ideogram in the user variable for use by VOCABULARY.

Comment: The implementation produces the correct run-time procedure according to the function described in the 79S. Note that the implementation has been changed from that >IN early versions of MVP-FORTH. Chaining of vocabularies is not possible with this implementation.

<VOCABULARYFIG>

MVP-FORTH

The run-time routine for a defining word to create a vocabulary definition <name>. Subsequent use of <name> will make it the CONTEXT vocabulary which is searched first by INTERPRET. The sequence “<name> DEFINITIONS” will also make <name> the CURRENT vocabulary into which new definitions are placed. In fig-Forth, <name> will be so chained as to include all definitions of the vocabulary in which <name> is itself defined. All vocabularies ultimately chain to Forth. By convention, vocabulary names are to be declared IMMEDIATE. See VOC-LINK. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Form: <VOCABULARYFIG> <name>

Implementations:

MVP: 8088/86:

```
: <VOCABULARYFIG>
  CREATE A081 , CURRENT @ CFA ,
  HERE VOC-LINK @ , VOC-LINK !
DOES> 2+ CONTEXT ! ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
' <VOCABULARYFIG> CFA 'VOCABULARY !
```

The example will change the function of VOCABULARY in the distribu-

<WORD>

tion version of MVP-FORTH to that used by fig-Forth. This is not allowed in the 79S.

Comment: The function of VOCABULARY defined by the 79S is restrictive. Many users have found the fig-Forth implementation much better. Other have developed other functions for VOCABULARY. By vectoring VOCABULARY, it is possible for a user to use a function of his own design as well.

<WORD>

char --- addr

MVP-FORTH

Receive characters from the input stream until the non-zero delimiting character is encountered or the input stream is exhausted, ignoring leading delimiters. The characters are stored as a packed string with the character count in the first character position. The actual delimiter encountered (char or null) is stored at the end of the text but not included in the count. If the input stream was exhausted as WORD is called, then a zero length will result. The address of the beginning of the packed string is left on the stack. (*Fig*) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: <WORD>
  'STREAM SWAP ENCLOSE DDUP >
  IF DDROP DDROP 0 HERE !
  ELSE >IN +! OVER - DUP >R
    HERE C! + HERE 1+ R> DUP 00FF >
    ABORT" INPUT 255" 1+ CMOVE
  THEN HERE ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: INPUT . " Input an integer -- "
  QUERY BL <WORD> NUMBER DROP ;
```

The definition provides for a prompt and then a pause for the operator to input the requested integer. Then the input character stream is

parsed with this ideogram, converted to a double precision value and reduced to a single precision value left on the stack.

Comment: This ideogram is vectored from WORD via user variable 'WORD'. In MVP-FORTH, WORD leaves the address of the initial count on top of the stack. As in fig-Forth, the string is stored at HERE.

= n1 n2 --- flag MVP-FORTH

True if n1 is equal to n2. (*Fig*) Leave a true flag if n1=n2; otherwise leave a false flag. (79S) Same as MVP. (83S) flag is true if n is equal to n. (F83) Returns true if the two elements on the stack are equal, False otherwise. (F-PC) Same as F83.

Pronounced: equals

Implementations:

MVP: 8088/86:

```
: = - NOT ;
```

Fig: 8088/86:

```
: = - 0= ;
```

F83: 8088/86:

```
CODE =
  AX POP    BX POP    AX BX CMP    YES JE
  NO #) JMP END-CODE
```

F-PC:

```
CODE =
  POP AX    POP CX    SUB AX, CX
  SUB AX, # 1    SBB AX, AX
  1 PUSH    END-CODE
```

Example:

```
45 45 =
```

The two values are placed on the stack and the test made. In this case the values are equal and a 1 is left on the stack as a true flag.

Comment: A common operator to all versions of Forth. The comparison destroys both comparands.

> n1 n2 --- flag MVP-FORTH

True if n1 is greater than n2. (*Fig*) Leave a true flag if n1 is greater than n2; otherwise a false flag. (79S) Same as MVP. (83S) Flag is true

>BINARY

if n1 is greater than n2. -32768 32767 > must return false; -32768 0 > must return false (F83) Compare the top two elements on the stack as signed integers and return true if n1 > n2. (F-PC) Same as F83.

Pronounced: greater-than

Implementations:

MVP: 8088/86:

```
: >   SWAP   <   ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE >
  AX POP    BX POP    AX BX CMP
  YES JG    NO #) JMP END-CODE
```

F-PC:

```
CODE >
  POP AX    POP BX    CMP BX, AX    <=
  IF  SUB AX, AX    1PUSH
  THEN  MOV AX, # TRUE  1PUSH    END-CODE
```

Example:

```
45 46 >
```

The two values are placed on the stack and the test is made. In this case the test is false and a 0 flag is left on the stack.

Comment: A common logical operator in all versions of Forth. The comparison destroys both comparands.

>BINARY

MVP-FORTH SUPPLEMENTAL

```
d1 addr1 --- d2 addr2
```

Same as CONVERT. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Pronounced: to-binary

Implementations:

MVP: 8088/86: Could be defined as:

```
: >BINARY   CONVERT   ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: TEST  BL  WORD  0  0  ROT  >BINARY  DROP  ;
```

A definition which provides a simple demonstration of the conversion of a WORD to a double precision binary value.

Comment: Some Forth implementations hash the name ideograms in order to save space. In so doing the 79S CONVERT, normally used, may conflict with CONTEXT, therefore the need of an alias.

>BODY

NOT USED IN MVP-FORTH

addr1 --- addr2

(Fig) Not defined. (79S) Not defined. (83S) addr2 is the parameter field address corresponding to the compilation address addr1. (F83) Go from code field to body. (F-PC) Go from code field address cfa to body address.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: >BODY  2+  ;
```

F-PC:

```
: >BODY  3 +  ;
```

Example:

```
' >BODY  >BODY  10  DUMP
```

Find the cfa of >BODY, go to the body address and dump the first 10 values.

Comment: This ideogram was introduced in a special vocabulary of the 83S. The offset is different in F83 and F-PC.

>LINK

>IN

--- addr

MVP-FORTH

Leave the address of a variable which contains the present character offset within the input stream. 0..1023 (*Fig*) Not defined. (79S) Same as MVP. (83S) The address of a variable which contains the present character offset within the input stream {{ 0..the number of characters in the input stream}}. See:WORD (*F83*) Number of characters interpreted so far. (*F-PC*) Same as F83.

Pronounced: to-in

Implementations:

MVP: 8088/86:

HEX

36 USER >IN

Fig: 8088/86:

Not used.

F83: 8088/86:

VARIABLE >IN (OFFSET INTO INPUT STREAM)

F-PC:

Same as F83.

Example:

```
: QUERY      TIB  @  50  EXPECT  0  >IN  !  ;
```

The value of the user variable, >IN, is set to 0 after EXPECT.

Comment: This ideogram replaces >IN in the earlier fig-Forth. Together with BLK, it determines the location of the next character from the input stream.

>LINK

NOT USED IN MVP-FORTH

addr1 --- addr2

(Fig) Not defined. (79S) Not defined. (83S *Definition Field Address Conversion Operators*) addr2 is the link field address corresponding to the compilation address addr1. (F83) Go from code field to link field. (F-PC) Go from code field address cfa to link field address lfa.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

: >LINK >NAME N>LINK ;

F-PC:

Same as F83.

Example:

' >LINK >LINK @ U.

Find the cfa of >LINK, move to the lfa, get its value and print it.

Comment: The ideogram was introduced in a special vocabulary of the 83S. The high level definition in both F83 and F-PC is the same.

>MARK

--- addr

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S *System Extension Word Set Glossary*) Used at the source of a forward branch. Typically used after either BRANCH or ?BRANCH. Compiles space in the dictionary for a branch address which will later be resolved by >RESOLVE . (F83) Set up for a Forward Branch. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

: >MARK HERE 0 , ;

F-PC:

: >MARK XHERE NIP 0 X, ;

Example:

>MARK

Used in compiling to allocate space for a forward reference.

Comment: The ideogram is part of the 83S *System Extension Word Set*. It helps take the burden off the programmer in keeping the pointers straight for forward referencing.

>NAME

NOT USED IN MVP-FORTH

>R

addr1 --- addr2

(Fig) Not defined. (79S) Not defined. (83S *Definition Field Address Conversion Operators*) addr2 is the name field address corresponding to the compilation address addr1. (F83) Go from code field to name field. (F-PC) Go from code field address cfa to name field address nfa.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE >NAME

```
DX POP    DP-H #) BX MOV    BX DEC
HSEG #) AX MOV    AX ES MOV
BEGIN    ES: 0 [BX] AL MOV    AH AH SUB    AL BX SUB
3 # BX SUB    ES: 0 [BX] DX CMP    0=
IF    BX INC    BX INC    BX PUSH    NEXT,    THEN
5 # BX SUB    0=
UNTIL    AX AX XOR    AX PUSH    NEXT
```

F-PC:

: >NAME

DUP CNHASH DUP Y@ SWAP 2+ Y@ (cfa sya mxya)

CNSRCH

IF DROP (LIT) [ROT , -X] THEN ;

Example:

' >NAME >NAME .ID

Get the code field address of >NAME, go to its name field address and print it.

Note: Some dialects of Forth use ID. rather than .ID.

Comment: This ideogram is part of a special vocabulary in the 83S. It can be used in writing a disassembly or decompile program.

>R

n ---

MVP-FORTH

Transfer n to the return stack. Every >R must be balanced by a R> in the same control structure nesting level of a colon-definition. (Fig) Remove a number from the computation stack and place as the most accessible on the return stack. Use should be balanced with R> in the same definition. (79S) Same as MVP. (83S) Transfers 16b to the

return stack. See: "9.3 Return Stack". (F83) Pops a value off of the parameter stack and pushes it onto return stack. It is dangerous to use this randomly! (F-PC) Same as F83.

Pronounced: to-r

Implementations:

MVP: 8088/86:

```
CODE >R    BX POP    BP DEC    BP DEC
          BX [BP], MOV    NEXT JMP    END-CODE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE >R
  AX POP    RP DEC    RP DEC    AX SS: 0 [RP] MOV    NEXT
```

F-PC:

```
CODE >R
  SUB RP, # 2          \ 4
  POP 0 [RP]           \ 22 = 26 cycles
  NEXT    END-CODE
```

Example:

```
: TEST    45    47    >R    1+    R>    ;
```

A contrived example: place two values on the stack and then increment the value immediately below the top of the stack.

Comment: CAUTION! This ideogram must be used with care to avoid crashing the system. Within its limitations, it is useful for accessing buried numbers. PICK and ROLL, however, offer a less dangerous alternative.

>RESOLVE

addr ---

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S System Extension Word Set Glossary) Used at the destination of a forward branch. Calculates the branch address (to the current location in the dictionary) using addr and places this branch address into the space left by >MARK. (F83) Resolve a Forward Branch. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

>TYPE

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: >RESOLVE  HERE  SWAP  !    ;
```

F-PC:

```
: >RESOLVE  XHERE  -ROT  SWAP  !L    ;
```

Example:

```
: ?RESOLVE  SWAP ?CONDITION  >RESOLVE  ;
```

The example resolved a forward branch with error checking in F-PC.

Comment: The ideogram is part of the 83S *System Extension Word Set*. It helps take the burden off the programmer in keeping the pointers straight for forward referencing.

>TYPE

addr n ---

MVP-FORTH SUPPLEMENTAL

Same as TYPE except that the output string is moved to the pad prior to output. Used in multiprogrammed systems to output strings from disk blocks. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) TYPE for multitasking systems. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

```
: >TYPE  ." USED IN MULTIPROGRAMMED SYSTEMS ONLY. "
IMMEDIATE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: >TYPE  TUCK PAD SWAP CMOVE  PAD SWAP TYPE  ;
```

F-PC:

Same as F83.

Example:

```
PAD  COUNT  >TYPE
```

In this implementation you are informed that the ideogram is not available for use.

Comment: This ideogram needs to be implemented only in multiuser systems.

>VIEW cfa --- vfa NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Go from code field to view field. (F-PC) Go from code field address cfa to view field address.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

: >VIEW >LINK 2+ ;

F-PC:

: >VIEW >LINK 2- ;

Example:

```
: FORGET
  BL WORD ?UPPERCASE DUP CURRENT @ HASH @
  (FIND) 0= ?MISSING DUP >VIEW (FORGET) ;
```

The example is from the implementation of FORGET in F-PC.

Comment: Those implementations which provide a view function need to find the location of the code field in the header structure.

? addr --- MVP-FORTH

Display the number at address, using the format of “ . ”. (Fig) Print the value contained at the address in free format according to the current base. (79S) Same as MVP. (83S) Not defined. (F83) Not defined. (F-PC) Display the contents of the 16-bit memory location addr as a signed value.

Pronounced: question-mark

Implementations:

MVP: 8088/86:

: ? @ . ;

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

: ? @ . ;

?BRANCH

F-PC:

Same as F83.

Example:

```
HEX 4AC2 ? DECIMAL
```

Place an address on the stack, and the value contained in the sixteen bits beginning with that address is printed.

Comment: Provides a quick way of finding the current value at a given address anywhere in memory. It is most useful in developing and debugging programs.

?BRANCH

flag ---

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S *System Extension Word Set Glossary*) When used in the form: `COMPILE ?BRANCH` a conditional branch operation is compiled. See `BRANCH`. When flag is true execution continues at the compilation address immediately following the branch address. (F83) Performs a conditional branch. If the top of the parameter stack is True, take the branch. If not, skip over the branch address which is in line. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
CODE ?BRANCH
```

```
  AX POP    AX AX OR   BRAN1 JE    IP INC    IP INC
NEXT
```

F-PC:

```
CODE ?BRANCH
```

```
  POP CX    CX<>0
```

```
  IF  ADD IP, # 2    NEXT    THEN
```

```
  MOV ES: IP, 0 [IP]  NEXT    END-CODE
```

Example:

```
: IF    COMPILE ?BRANCH  ?>MARK ;    IMMEDIATE
```

The example is taken from the implementation of IF in F-PC.

Comment: The ideogram is a replacement of 0BRANCH in older im-

plementations of Forth. The change was made in a system vocabulary in the 83S.

?COMP

MVP-FORTH

Issue an error message if not compiling. (*Fig*) Issue error message if not compiling. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Abort if we are not compiling.

Implementations:

MVP: 8088/86:

```
: ?COMP    STATE @    NOT
  ABORT"    COMPILE ONLY "    ;
```

Fig: 8088/86:

```
HEX
: ?COMP    STATE @    0=    11    ?ERROR ;
```

F83: 8088/86:

Not used.

F-PC:

```
: ?COMP
  STATE @ 0= ABORT" Use only while compiling" ;
```

Example:

```
: BEGIN    ?COMP    HERE    1    ;
```

Ensure that you are compiling as in this definition.

Comment: An ideogram used by fig-Forth and some other implementations. It is an error-handling function which, in MVP-FORTH, types its message from an in-line ABORT" string.

?CONFIGURE

MVP-FORTH

Display the current configuration for all available disk drives. The density code for each drive is given as an integer from 0 through 6, with the following interpretation:

```
0 - 5" Single Sided, Single Density (5-SSSD)
1 - 5" Double Sided, Single Density (8-SSSD)
2 - 8" Double Sided, Single Density (8-DSSD)
3 - 8" Single Sided, Double Density (8-SSDD)
4 - 8" Double Sided, Double Density (8-DSDD)
5 - 8" Single Sided, Extended Density (8-SSEXT)
6 - 8" Double Sided, Extended Density (8-DSEXT)
```

?CONFIGURE

The number of drives available is determined by the value of the constant MAX-DRV, which may be altered. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: ?CONFIGURE    CR CR MAX-DRV .  
    ." DRIVES WITH DENSITIES: "  
    MAX-DRV 0 DO I DR-DEN 2 SPACES . LOOP  
    CR CR ." DENSITY CODE "  
    CR ." 0 - 5-SSSD"    CR ." 1 - 5-DSSD"  
    CR ." 2 - 8-DSSD"    CR ." 3 - 8-SSDD"  
    CR ." 4 - 8-DSDD"    CR ." 5 - 8-SSEXT"  
    CR ." 6 - 8-DSEXT"  
    CR ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

?CONFIGURE

Display current physical drive parameters, without changing their settings. Stacks and dictionary remain unaffected.

Comment: ?CONFIGURE displays the contents of the array, DEN, for drives numbered 0 through MAX-DRV less one. To change these values from the terminal, use CONFIGURE. The density codes refer to the order of items in the arrays SEC/TR and BLK/DRV. They can be easily examined with DUMP and modified as necessary. The prompts can be patched for a particular installation. (See *The Forth Guide*) I have made mine:

```
0 160K DSK  
1 320K DSK  
2 360K DSK  
3 1.2M DSK  
4 RAM-DISK  
5 HARD-DISK  
6 NULL-DISK
```

?CSP

MVP-FORTH

Issue error message if stack position differs from value saved in CSP. (*Fig*) Issue error message if stack position differs from value saved in CSP. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Issue error message if stack has changed. (*F-PC*) Issue error message if stack has changed since the most recent !CSP.

Implementations:

MVP: 8088/86:

```
: ?CSP SP@ CSP @ -
  ABORT" DEFINITION NOT FINISHED " ;
```

Fig: 8088/86:

HEX

```
: ?CSP SP@ CSP @ - 14 ?ERROR ;
```

F83: 8088/86:

```
: ?CSP SP@ CSP @ <> ABORT" Stack Changed" ;
```

F-PC:

Same as F83.

Example:

```
: ; ?CSP COMPILE ;S SMUDGE [COMPILE] [ ;
IMMEDIATE
```

In this definition from the MVP-FORTH source code, the ideogram does the error checking.

Comment: This ideogram is used by fig-Forth and some other implementations. It is an error-handling function which, in MVP-FORTH, types its message from an in-line ABORT" string.

?DO

limit start ---

NOT USED IN MVP-FORTH

(*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Same as DO except that the loop will not be executed if start = limit.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

?DUP

F83: 8088/86:

```
: ?DO    COMPILE (?DO)  >MARK 3000  ; IMMEDIATE
```

F-PC:

```
: ?DO    COMPILE (?DO)  ?>MARK  ; IMMEDIATE
```

Example:

```
: TEST    40  40  ?DO  I    U.  LOOP  ;
```

Executing TEST will do nothing.

Comment: If one were to use DO, the loop would be executed 64K times.

?DUP

n --- n (n)

MVP-FORTH

Duplicate n if it is non-zero. (*Fig*) Not defined. (*79S*) Same as MVP. (*83S*) Duplicate 16b if it is non-zero. (*F83*) Duplicate the top of the stack if it is non-zero. (*F-PC*) Same as F83.

Pronounced: query-dup

Implementations:

MVP: 8088/86:

```
: ?DUP  DUP  IF  DUP  THEN  ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
CODE ?DUP
```

```
    AX POP    AX AX OR    0<> IF    AX PUSH    THEN    1PUSH
```

F-PC:

```
CODE ?DUP
```

```
    MOV DI, SP    MOV CX, 0 [D]    CX<>0  
    IF    PUSH CX    THEN    NEXT    END-CODE
```

Example:

```
: INFORM  DISK-ERROR  @  ?DUP  
    IF  ."  DISK  ERROR  CODE  "  .  THEN  ;
```

The message is printed only when the value is non-zero.

Comment: This is the current ideogram for the now obsolete one -DUP. It saves having to DROP the value from the stack should a test, as with IF, turn out false.

?ERROR

f n ---

NOT USED IN MVP-FORTH

(Fig) Issue an error message number n, if the boolean flag is true. (79S) Not defined. (83S) Not defined. (F83) Note: uses different stack requirements: (addr len f ---). Maybe to indicate an error. Change this to alter ABORT". (F-PC) (Note: uses different stack requirements: a1 n1 f1 --- .) If f1 is true then display the error message a1, n1 and QUIT. Change this to alter ABORT".

Implementations:

MVP: 8088/86:

This ideogram is not implemented because the messages are in line and not handled from the disk as in fig-Forth.

Fig: 8088/86:

```
: ?ERROR  SWAP
  IF  ERROR  ELSE  DROP  ENDIF  ;
```

F83: 8088/86:

```
DEFER ?ERROR
: (?ERROR)
  IF >R >R SP0 @ SP! PRINTING OFF BLK @
    IF >IN @ BLK @ WHERE THEN
      R> R> SPACE TYPE SPACE QUIT
    ELSE 2DROP THEN ;
  ' (?ERROR) IS ?ERROR
```

F-PC:

```
DEFER ?ERROR
: (?ERROR)
  IF ['] <RUN> IS RUN ERRFIX 2>R SP0 @ SP!
    PRINTING OFF 2R> SPACE TYPE SPACE QUIT
  ELSE 2DROP THEN ;
  ' (?ERROR) IS ?ERROR
```

Comment: Used to select an error message from disk in fig-Forth if WARNING is set to 1, or the message number if WARNING is set to zero. Other implementations of Forth handle error messages in different ways.

?EXEC

NOT USED IN MVP-FORTH

(Fig) Issue an error message if not executing. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

?LOADING

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

HEX

: ?EXEC STAT @ 12 ?ERROR ;

F83: 8088/86:

Not used.

F-PC:

: ?EXEC STATE @

ABORT" Can not be used while Compiling!" ;

Comment: Not a 79S ideogram. Used in fig-Forth and may be present in other implementations of Forth.

?LOADING

MVP-FORTH

Issue an error message if not loading. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

: ?LOADING BLK @ NOT
ABORT" LOADING ONLY " ;

Fig: 8088/86:

HEX

: ?LOADING BLK @ 0= 16 ?ERROR ;

F83: 8088/86:

Not used.

F-PC:

: ?LOADING LOADING @ 0=
IF CRUNSAVE IS RUN
TRUE ABORT" Use ONLY while loading"
THEN ;

Example:

: -->
?LOADING 0 >IN ! B/SCR @ OVER
MOD - BLK +! ; IMMEDIATE

This fig-Forth definition illustrates the use of the ideogram. It will not

work in MVP-FORTH, and there are better ways to load multiple screens (See THRU).

Comment: Not a 79S ideogram. Used in fig-Forth and may be present in other implementations of Forth.

?PAIRS

n1 n2 ---

MVP-FORTH

Issue an error message if n1 does not equal n2. The message indicates that compiled conditionals do not match. (*Fig*) Same as MVP. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Simple compile time error checking. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: ?PAIRS -  
  ABORT" CONDITIONALS NOT PAIRED " ;
```

Fig: 8088/86:

```
HEX  
: ?PAIRS - 13 ?ERROR ;
```

F83: 8088/86:

```
: ?PAIRS <> ABORT" Conditionals Wrong" ;
```

F-PC:

Not used.

Example:

```
: AGAIN 1 ?PAIRS  
  COMPILE BRANCH HERE - , ;
```

In a correct colon definition, " 1 ?PAIRS " will encounter the value 1 left on the stack by the corresponding BEGIN. This ensures the proper nesting in practically all cases.

Comment: Not a 79S ideogram. Used in fig-Forth and may be present in other implementations of Forth.

?STACK

MVP-FORTH

Issue an error message if the stack is out of bounds. This definition may be installation dependent. (*Fig*) Same as MVP. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Check for parameter stack underflow or overflow and issue appropriate error message if detected. (*F-PC*) Same as F83.

?STACK

Implementations:

MVP: 8088/86:

```
: ?STACK  SP@ 2+ S0 SWAP U< <ABORT"> STACK
OUT OF BOUNDS " SP@ HERE 80 + U<R
<ABORT"> FULL STACK " ;
```

Fig: 8088/86:

```
HEX
: ?STACK
SP@ S0 @ SWAP U< 1 ?ERROR
SP@ HERE 80 + U< 7 ?ERROR ;
```

F83: 8088/86:

```
: ?STACK
SP@ SP0 @ SWAP U< ABORT" Stack Underflow"
SP@ PAD U< ABORT" Stack Overflow" ;
```

F-PC:

```
CODE (?STACK)
MOV CX, SP    MOV BX, UP
MOV DX, SP0 [BX]  CMP DX, CX    U<
IF  MOV AX, # ' STACKUNDER  JMP AX
THEN  MOV DX, DP [BX]
ADD DX, # 80    CMP CX, DX    U<
IF  MOV AX, # ' STACKOVER  JMP AX
THEN  ADD DX, # 200    CMP CX, DX    U<
IF  MOV AX, # ' WARNOVER  JMP AX
THEN  NEXT  END-CODE
DEFER ?STACK    ' (?STACK) IS ?STACK
```

Example:

```
: INTERPRET
BEGIN -FIND
IF STATE @
IF 2- , ELSE 2- EXECUTE THEN ?STACK
ELSE HERE NUMBER DPL @ 1+
IF [COMPILE] DLITERAL
ELSE DROP [COMPILE] LITERAL
THEN ?STACK
THEN
AGAIN ;
```

This definition from an early version of the source illustrates this ideogram. Note that stack checking is an expensive operation best

reserved for outermost loops and recursive definitions.

Comment: Not a 79S ideogram. Used in fig-Forth and may be present in other implementations of Forth. *Starting Forth* uses this ideogram with a different function.

?STREAM

f ---

MVP-FORTH

Issue an error message if the flag is true indicating that the input stream is exhausted. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: ?STREAM  ABORT" INPUT STREAM EXHAUSTED" ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: (  -1  >IN  +!  29  WORD  C@  1+  
  HERE  +  C@  29  =  NOT  
  ?STREAM  ;  IMMEDIATE
```

The example comes from the MVP-FORTH source code.

Comment: This ideogram signals an error condition, and is available for the programmer.

?TERMINAL

--- f

MVP-FORTH

Perform a test of the terminal keyboard for actuation of the break key. A true flag indicates actuation. This definition is installation dependent. (*Fig*) Same as MVP. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: ?TERMINAL  '?TERMINAL  @  EXECUTE  ;
```

Fig: 8088/86:

@

CODE ?TERMINAL PQTER JMP

NOTE: This is for a CPM86 system.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: TEST 1000 1 DO I . ?TERMINAL
  IF LEAVE THEN LOOP ;
```

With this definition, TEST , will start printing numbers which can be interrupted any time by hitting any key in the MVP-FORTH implementation.

Comment: Not a 79S ideogram. Used in fig-Forth and may be present in other implementations of Forth. It is a convenient test in a definition which permits interrupting a loop by entering something from the keyboard. There is nothing like it when a paper in the printer hangs up. It is, however not defined in most versions of Forth, because of its installation dependency which would limit portability. However, it is usually a simple matter to modify a program according to the installation requirements. In the IBM implementation, a true flag is the ASCII value of the character struck.

@

addr --- n

MVP-FORTH

Leave on the stack the number contained at addr. (*Fig*) Leave the 16-bit contents of address. (*79S*) Same as MVP. (*83S*) 16b is the value at addr. (*F83*) Fetch a 16-bit value from addr. (*F-PC*) Same as F83.

Pronounced: fetch

Implementations:

MVP: 8088/86:

```
CODE @
  BX POP [BX] AX, MOV APUSH JMP END-CODE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE @
  BX POP 0 [BX] PUSH NEXT
```

F-PC:

Same as F83.

Example:

```
HEX 4AC2 @ . DECIMAL
```

Place the address on the stack, fetch the value at that address and then print it.

Comment: A common and frequently used Forth ideogram present in almost all implementations.

@L seg addr --- n MVP-FORTH

In an 8086/8088 system, leave on the stack the number contained in seg at addr. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. Uses L@. (*F83*) Not defined. (*F-PC*) - load word long from seg and adr.

Implementations:

MVP: 8088/86:

```
CODE @L
  BX POP DS POP [BX] AX, MOV
  CS BX, MOV BX DS, MOV APUSH JMP END-CODE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

```
CODE @L
  POP BX POP DS MOV AX, 0 [BX]
  MOV BX, CS MOV DS, BX 1 PUSH END-CODE
```

Example:

```
HEX
40 17 @L .
DECIMAL
```

The example examines the "Caps Lock" key flag on the IBM Personal Computer and print its value.

Comment: The function of this ideogram allows one to examine parts of memory outside the segments used by Forth.

ABORT"

ABORT

MVP-FORTH

Clear the data and return stacks, setting execution mode. Return control to the terminal. (*Fig*) Clear the stacks and enter the execution state. Return control to the operators terminal, printing a message appropriate to the installation. (79S) Same as MVP. (83S) Clears the data stack and performs the function of QUIT. No message is displayed. (F83) Stop the system and indicate an error. (F-PC) Clear the data stack and QUIT; no message is displayed.

Implementations:

MVP: 8088/86:

```
: ABORT
  'ABORT @ EXECUTE ;
```

Fig: 8088/86:

```
HEX
: ABORT
  SP! DECIMAL ?STACK CR .CPU ." Fig-Forth "
  FIGREL+30H C, ADOT C, FIGREV+30H C,
  FORTH DEFINITIONS QUIT ;
```

F83: 8088/86:

```
: ABORT  SP0 @ SP!  QUIT  ;
```

F-PC:

```
: ABORT  TRUE ABORT" "  ;
```

Example:

```
ABORT
```

This simple ideogram will clear the stack and return to Forth so that you can begin again. Your definitions are not forgotten.

Comment: In some versions of Forth, this ideogram also includes a statement of the name of the version and other information, while in 79S, no indication is made. Since it is vectored, you may modify its actions however you please. Normally, ABORT invokes <ABORT>.

ABORT"

flag ---

MVP-FORTH

Used in a colon-definition. If the flag is true, print the following text, till " . Then execute ABORT. (*Fig*) Not defined. (79S *Reference Word Set*) Same as MVP. (83S) When later executed, if flag is true the characters ccc, delimited by " (close-quote), are displayed and then a system dependent error abort sequence, including the function of

ABORT, is performed. If flag is false, the flag is dropped and execution continues. The blank following ABORT" is not part of ccc. (F83) If the flag is true, issue an error message and quit. (F-PC) If the flag is true, issue <message> and ABORT.

Form: ABORT" stack empty"

Pronounced: abort-quote

Implementations:

MVP: 8088/86:

```
: ABORT"
  ?COMP  COMPIL ABORT"  'STREAM      C@  22  =
  IF  1  >IN  +!  0  C,
  ELSE  22 WORD  DUP  C@  1+  SWAP  OVER
    +  C@  22  =  NOT  ?STREAM  ALLOT
  THEN  ;  IMMEDIATE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: ABORT"  COMPIL (ABORT")  , " ;  IMMEDIATE
```

F-PC:

```
: ABORT:  COMPIL (ABORT")  X, " ; IMMEDIATE
```

Example:

```
: ?/
  DUP  0=  ABORT" DIVISION BY ZERO "  /  ;
```

This example invents a new division operator which works exactly like / unless the divisor is zero.

Comment: A convenient way of providing error messages when aborting a routine.

ABS

n1 --- n2

MVP-FORTH

Leave the absolute value of a number. (Fig) Leave the absolute value of n as u. (79S Reference Word Set) Same as MVP. (83S) u is the absolute value of n. If n is -32,768 then u is the same value. See: "arithmetic, two's complement". (F83) Return the absolute value of the 16-bit integer on the stack. (F-PC) Same as F83.

Pronounced: absolute

AGAIN

Implementations:

MVP: 8088/86:

```
: ABS
  DUP  + -  ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE ABS
  AX POP      CWD      DX AX XOR      DX AX SUB      1PUSH
```

F-PC:

Same as F83.

Example:

```
-45  ABS
```

Enter the negative value on the stack and it is converted to a positive value.

Comment: A common ideogram in all versions of Forth, operates on signed single precision integers. Note that “-32768 ABS” returns the negative value - 32768 because its absolute value cannot be represented as a signed 16-bit number.

AGAIN

MVP-FORTH

```
addr n --- ( compiling )
          --- ( run-time )
```

Effect an unconditional jump back to the start of a BEGIN-AGAIN loop. (*Fig*) At run-time, AGAIN forces execution to return the corresponding BEGIN. There is no effect on the stack. Execution cannot leave this loop (unless R> DROP is executed one level below). At compile time, AGAIN compiles BRANCH with an offset from HERE to addr. n is used for compile-time error checking. (*79S Reserved Word Set*) Same as MVP. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Unconditional jump to just after BEGIN in a BEGIN ... AGAIN loop.

Implementations:

MVP: 8088/86:

```
: AGAIN  1  ?PAIRS  COMPILE  BRANCH
  HERE  -  ,  ;  IMMEDIATE
```

Fig: 8088/86:

```
: AGAIN
```

```
1    ?PAIRS  COMPILE  BRANCH  BACK ;  IMMEDIATE
```

F83: 8088/86:

```
: AGAIN
  COMPILE  BRANCH  2000 ?PAIRS  <RESOLVE ; IMMEDIATE
```

F-PC:

```
: AGAIN
  COMPILE  DOAGAIN  ?<RESOLVE  ; IMMEDIATE
```

Example:

```
: TEST
  0 BEGIN  1+  DUP  .  AGAIN  ;
```

You will have to reboot your system if you try this example — a good lesson ! Defines an infinite loop to print the number series beginning with the value 1.

Comment: An ideogram used to implement MVP-FORTH, but not part of 79S.

ALLOT

n ---

MVP-FORTH

Add n bytes to the parameter field of the most recently defined word. (*Fig*) Add the signed number to the dictionary pointer DP. May be used to reserve dictionary space or re-originate memory. n is with regard to computer address type (byte or word). (79S) Same as MVP. (83S) Allocates n bytes in the dictionary. The address of the next available dictionary location is updated accordingly. (F83) Allocate more space in the dictionary. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

```
: ALLOT  DP  +!  ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Same as MVP.

F-PC:

Same as MVP.

Example:

```
VARIABLE  NEW-VALUE  20  ALLOT
```

After creating an uninitialized variable, NEW-VALUE, 20 bytes are

AND

skipped over in the dictionary before a new definition will be compiled. These could be used as a small buffer, for example.

Comment: Allows one to reserve space in the definition of a new ideogram. This space can be used in a variety of ways such as making room for an array. It is the normal way to reserve space for arrays, buffers and other data structures. In this implementation, there is no security. If one were to ALLOT into the data stack, he could crash the system.

ALSO

NOT USED IN MVP-FORTH

(*Fig*) Not defined. (79*S*) Not defined. (83*S*) Not defined (F83) Not defined. (F-PC) Duplicate the top of the CONTEXT vocabulary stack. (Ragsdale) The transient vocabulary becomes the first vocabulary in the resident portion of the search order. Up to the last two resident vocabularies will also be preserved, in order, forming the resident search order.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

: ALSO CONTEXT DUP 2+ #VOCS 2- 2* CMOVE>

F-PC:

Same as F83.

Example:

ONLY FORTH ALSO ASSEMBLER

Make FORTH the root vocabulary and add the ASSEMBLER. The ASSEMBLER vocabulary will be searched first.

Comment: This ideogram is a part of an alternative to the fig-Forth and 78*S* vocabulary structures. It was introduced by William F. Ragsdale at the 1982 FORML Conference. A number of implementations have been adopted.

AND

n1 n2 --- n3

MVP-FORTH

Leave the bitwise logical 'and' of n1 and n2. (*Fig*) Leave the bitwise logical 'and' of n1 and n2 as n3. (79*S*) Same as MVP. (83*S*) 16b3 is

the bit-by-bit logical 'and' of 16b1 with 16b2. (*F83*) Returns the bitwise AND of n1 and n2 on the stack. (*F-PC*) Same as *F83*.

Implementations:

MVP: 8088/86:

```
CODE AND
  BX POP    AX POP    BX AX, AND    APUSH JMP    END-CODE
```

Fig: 8088/86:

```
CODE AND
  AX POP    BX POP    BX AX, AND    APUSH JMP
```

F83: 8088/86:

```
CODE AND
  BX POP    AX POP    BX AX AND    1PUSH
```

F-PC:

Same as *F83*.

Example:

```
HEX
: TEST
  E5 P@ 20 AND IF ." READY" THEN ;
```

This ideogram is used as a mask to see if bit 6 of an input port is set and, if it is, to type READY.

Comment: This ideogram allows the masking of the bit pattern of one number on the stack with the other, an operation which is common in computing.

APUSH

--- addr

MVP-FORTH

A constant used in 8086/8088 implementations pointing to a machine code entry point which pushes the contents of the AX register onto the stack and then falls through to NEXT, the inner interpreter. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
APUSH CONSTANT APUSH
```

Fig: 8088/86:

Not used.

F83: 8088/86:

ASCII

Not defined.

F-PC:

Not defined.

Example:

```
CODE +  
  AX POP  BX POP  AX, BX ADD  APUSH JMP  END-CODE
```

This example is taken from the MVP-FORTH 8086/8088 source code.

Comment: This is a machine, system, and an implementation dependent location used only at the machine code level. It allows trimming one byte off CODE definitions which would otherwise end with AX PUSH NEXT JMP END-CODE.

ASSEMBLER

NOT USED IN MVP-FORTH

Select assembler as the CONTEXT vocabulary. (*Fig*) Not defined. (*79S Assembler Word Set*) MVP. (*83S Assembler Extension Word Set*) Execution replaces the first vocabulary in the search order with the ASSEMBLER vocabulary. See: VOCUABULARY (*F83*) Define the vocabulary to be filled later. (*F-PC*) - The VOCABULARY that contains all of the assembler definitions.

Comment: The implementation of this vocabulary is left to the user. It is machine- and implementation- dependent. Although not a part of the MVP-FORTH implementation, it may be written and loaded in a high-level source code form. VOCABULARY ASSEMBLER

ASCII

| char --- n

NOT USED IN MVP-FORTH

(*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Compile the next character as a literal. (*F-PC*) Compile the next character in the input stream as a literal ASCII integer.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
CODE (ASCII)  ( -- n )  AX LODS  1PUSH  
: ASCII      BL WORD  1+ C@  STATE @  
  IF COMPILE (ASCII) ,  THEN  ; IMMEDIATE
```

F-PC:

```
: ASCII    BL WORD    1+ C@    STATE @  
  IF [COMPILE] LITERAL THEN ; IMMEDIATE
```

Example:

```
ASCII    ABC    U.
```

The ASCII value of the first character in the string ABC , 65, is printed on the screen.

Comment: The ideogram is a great convenience if the programmer does not remember the value of an ASCII character. It is also a help to code a value this way if at a later time you will not recognize a character by the ASCII value.

B/BUF

--- n

NOT USED IN MVP-FORTH

A constant leaving 1024, the number of bytes per block buffer. (Not used in MVP 1.0405.03) (*Fig*) This constant leaves the number of bytes per disc buffer, the byte count read from disc by BLOCK. (*79S Reverence Word Set*) Same as MVP. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

```
KBBUF    CONSTANT    B/BUF
```

F83: 8088/86:

```
1024    CONSTANT    B/BUF
```

F-PC:

Not used.

Example:

```
B/BUF    .
```

This ideogram followed by . will print the current size of the disk buffers.

Comment: An installation dependent primitive which may be available in other implementations of Forth. In many implementations of fig-Forth, the value of this constant may be hex 80. Note that the implementation values are all in hexadecimal.

BASE

B/SCR

--- n

NOT USED IN MVP-FORTH

(Fig) This constant leaves the number of blocks per editing screen. By convention, an editing screen is 1024 bytes organized as 16 lines of 64 characters each. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

HEX

[400 KBBUF /] CONSTANT B/SCR

F83: 8088/86:

Not used.

F-PC:

Not used.

Comment: An installation dependent primitive necessary in some versions of Forth.

BACK

addr ---

NOT USED IN MVP-FORTH

(Fig) Calculate the backward branch offset from HERE to addr and compile into the next available dictionary memory address. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

: BACK HERE - , ;

F83: 8088/86:

Not used.

F-PC:

Not used.

Comment: Not used in the current implementations.

BASE

--- addr

MVP-FORTH

Leave the address of a variable containing the current input-output

numeric conversion base. 2..70 (*Fig*) A user variable containing the current number base used for input and output conversion. (79S) Same as MVP. (83S) The address of a variable containing the current numeric conversion radix. {{2..72}} (*F83*) The current numeric base for number input output. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

HEX 38 USER BASE

Fig: 8088/86:

HEX 26 USER BASE

F83: 8088/86:

VARIABLE BASE

F-PC:

Same as F83.

Example:

: BINARY 2 BASE ! ;

Add the definition of BINARY to your vocabulary.

Comment: The reference variable used to convert numeric input to the binary form in which it is stored on the stack and in memory. It can also be used for coding alphanumeric information by utilizing unusual values.

BDOS n1 fun --- n2 NOT USED IN MVP-FORTH

(*Fig*) Not defined. (79S) Not defined. (83S) Not defined. (*F83*) Load up the registers and do a DOS system call. Return the result placed in the A register on the stack. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE BDOS

AX POP AL AH MOV DX POP 21 INT AH AH SUP 1PUSH

F-PC:

CODE <BDOS>

BEGIN

```
POP AX    MOV AH, AL    POP DX
INT $21    SUB AH, AH
1PUSH      END-CODE
DEFER BDOS    ' <BDOS> IS BDOS
```

Example:

```
56 3 BDOS U.
```

Execute the DOS function 56 with a parameter of 2 and print the resulting code.

Comment: This function is the same as `SYSCALL` in MVP-FORTH. The DOS function 56 will test the format of the drive selected by the parameter beginning with Drive A having a parameter of 1.

BEEP

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Ring the bell on the terminal. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: BEEP    BELL EMIT ;
```

F-PC:

```
: %BEEP    BELL (EMIT) #OUT DECR ;
DEFER BEEP    /%BEEP IS BEEP
```

Example:

```
BEEP
```

The example will output a tone on the system's speaker

Comment: Adding a BEEP to the end of a long function will alert you to the completion of a process.

BEGIN

--- addr n (compile time)

MVP-FORTH

Used in a colon definition. BEGIN marks the start of a word sequence for repetitive execution. A BEGIN-UNTIL loop will be repeated until flag is true. A BEGIN-WHILE-REPEAT loop will be repeated until flag

is false. The ideograms after UNTIL or REPEAT will be executed when either loop is finished. flag is always dropped after being tested. (*Fig*) At run-time, BEGIN marks the start of a sequence that may be repetitively executed. It serves as a return point from the corresponding UNTIL, AGAIN or REPEAT. When executing UNTIL, a return to BEGIN will occur if the top of the stack is false; for AGAIN and REPEAT a return to BEGIN always occurs. At compile time BEGIN leaves its return address and n for compiler error checking. (*79S*) Same as MVP. (*83S*) BEGIN marks the start of a word sequence for repetitive execution. A BEGIN-UNTIL loop will be repeated until flag is true. A BEGIN-WHILE-REPEAT loop will be repeated until flag is false. The words after UNTIL or REPEAT will be executed when either loop is finished. sys is balanced with its corresponding UNTIL or WHILE . See: "9.9 Control Structures" (*F83*) Not defined. (*F-PC*) Form: as below.

Form: BEGIN ... flag UNTIL
 BEGIN ... flag WHILE ... REPEAT

Implementations:

MVP: 8088/86:

```
: BEGIN
  ?COMP HERE 1 ; IMMEDIATE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
: BEGIN
  <MARK 2000 ; IMMEDIATE
```

F-PC:

```
: BEGIN
  COMPILE DOBEGIN ?<MARK ; IMMEDIATE
```

Example:

```
: TEST
  0 BEGIN 1+ DUP . DUP 10 = UNTIL
  DROP ;
```

A definition which will print the values 1 through 10.

Comment: Used in colon definitions to delimit a nestable structure controlling repetitive execution.

BETWEEN

BELL --- n NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Name for a bell. (F-PC) Return the value 7, an ASCII BELL char.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

7 CONSTANT BELL

F-PC:

Same as F83.

Example:

BELL EMIT

The example will perform the function of BEEP; it will output a tone on the system's speaker

Comment: By defining BELL as a constant, it is not necessary to remember the value of the ASCII code.

BETWEEN NOT USED IN MVP-FORTH

n min max --- f

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Return true if min <= n1 <= max, otherwise false. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE BETWEEN

```
DX POP    BX POP    CX POP    AX AX SUB    CX DX CMP    0<
IF 1PUSH, THEN BX CX CMP 0<
IF 1PUSH, THEN AX DEC 1PUSH
```

F-PC:

CODE BETWEEN

```

XOR AX, AX    POP BX    POP CX
POP DX    CMP DX, BA    <=
IF    CMP DX, CX    >=
    IF    DC AX    THEN
THEN    1PUSH    END-CODE

```

Example:

```
10 0 100 BETWEEN
```

Test 10 to see if it is between 0 100. In the example it will leave a true flag on the stack.

Note that the test is inclusive. That is the value is equal to or between the limits. The only difference with **WITHIN** is that there the test is not inclusive with the upper limit.

Comment: Though not in any of the standards, this ideogram is commonly used. The high level Forth definition in F-PC can be added to most dialects where needed. However, for speed the code definition of F83 or F-PC may be useful.

BINARY

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) All subsequent numeric I-O will be in binary. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: BINARY    2 BASE ! ;
```

F-PC:

Not used.

Example:

```

: .B    BASE @    SWAP 0 SWAP 0 BINARY
    <# # # # # # # # # # SPACE HOLD # # # # # # # # # #>
    TYPE    BASE ! ;

```

Defines a word to print all 16-bits of an integer. The current base is saved, the integer moved to the correct place for the formatting of digit output, make the base binary, format and type the value and return to the original base.

BLANK

Comment: For those concerned with bit images, BINARY is easy to define. Perhaps one might want to format the output of a 16-bit value to show all bits as shown in the example.

```
BL      ---  C      MVP-FORTH
```

A constant that leaves the ASCII value for "blank". (*Fig*) Same as MVP. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Name for blank. (*F-PC*) Return hex 20, decimal 32 the value of an ASCII space.

Implementations:

MVP: 8088/86:

HEX 20 CONSTANT BL

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

32 CONSTANT BL

F-PC:

Same as F83.

Example:

```
: SPACE    BL    EMIT    ;
```

A definition which will print a space.

Comment: A useful ideogram allowing one to enter the ASCII value of a “blank” independent of the number system currently in use. Though not always included in implementations, it can be easily added if needed.

```

BLANK      addr n ---      MVP-FORTH

```

Fill an area of memory over n bytes with the value for ASCII blank, starting at addr. If n is less than or equal to zero, take no action. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Fill the string with blanks. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

: BLANK BL FILL ;

Fig: 8088/86:

Not used.

F83: 8088/86:

Same as MVP.

F-PC:

Same as MVP.

Example:

```
VARIABLE NAME 18 ALLOT
NAME 20 BLANK
```

Create a memory location of 20 bytes to store a name and then initialize the field with ASCII spaces.

Comment: This ideogram and its plural do exactly the same thing. There is no reason to have both available. This ideogram does not depend upon knowing the ASCII value of a blank in the current number base. fig-Forth gives this function the name BLANKS.

BLANKS addr n --- NOT USED IN MVP-FORTH

(Fig) Fill an area of memory beginning at addr with blanks. This ideogram, in fig-Forth, has the same function as BLANK in MVP-FORTH. (79S Reference Word Set) Fill an area of memory over n bytes with the value for ASCII blank, starting at addr. If n is less than or equal to zero, take no action. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

```
: BLANKS      BL FILL ;
```

F83: 8088/86:

Not used.

F-PC:

Not used.

Comment: This ideogram is now used in the singular form, BLANK.

BLK --- addr MVP-FORTH

Leave the address of a variable containing the number of the mass storage block being interpreted as the input stream. If the content is zero, the input stream is taken from the terminal. The value of the

BLK/DRV

variable is an unsigned number. (*Fig*) A user variable containing the block number being interpreted. If zero, input is being taken from the terminal input buffer. (79S) Same as MVP. (83S) The address of a variable containing the number of the mass storage block being interpreted as the input stream. If the value of BLK is zero the input stream is taken from the text input buffer. {{0..the number of blocks available -1}} See: TIB "input stream" (F83) If non-zero, the block number we are interpreting. (F-PC) Not defined.

Pronounced: b-l-k

Implementations:

MVP: 8088/86:

HEX 3A USER BLK

Fig: 8088/86:

HEX 16 USER BLK

F83: 8088/86:

VARIABLE BLK

F-PC:

Not used.

Example:

BLK @ .

When entered from the terminal, this user variable will print the value of 0 indicating that the terminal is being interpreted.

Comment: A common ideogram in most versions of Forth. Together with >IN, it determines the location of the next character from the input stream.

BLK/DRV

--- addr

MVP-FORTH DISK I-O

A variable beginning a seven item array containing the number of blocks of 1024 bytes on a drive of a given density format. (*Fig*) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

HEX VARIABLE BLK/DRV A0 BLK/DRV !
140 , 1F4 , 1F4 , 3E8 , 268 , 4D0 ,

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: BPDRV  DENSITY  @  6  MIN  2*  BLK/DRV  +  @  ;
```

The example is from the source code. Its function is to leave the number of blocks on a drive according to the present value of DENSITY.

Comment: An array used to calculate the actual drive to be accessed according to the block number being requested. Early implementations of MVP-FORTH used a similar array, SEC/DR, however, because of variations in formatting, BLK/DRV is more convenient. Note that the actual values in the array are system dependent. For the IBM Personal Computer with single sided 5 inch drives, the correct value of the first element of the array is 0A0 hex. (Each disk will hold 160 blocks.)

BLOCK

n --- addr

MVP-FORTH

Leave the address of the first byte in block n. If the block is not already in memory, it is transferred from mass storage into whichever memory buffer has been least recently accessed. If the block occupying that buffer has been UPDATED (i.e. modified), it is rewritten onto mass storage before block n is read into the buffer. n is an unsigned number. If correct mass storage read or write is not possible, an error condition exists. Only data within the latest block referenced by BLOCK is valid by byte address, due to sharing of the block buffers. (Fig) Leave the memory address of the block buffer containing block n. If the block is not already in memory, it is transferred from disc to whichever buffer was least recently written. If the block occupying that buffer has been marked as updated, it is rewritten to disc before block n is read into the buffer. See also BUFFER R/W UPDATE FLUSH (79S) Same as MVP. (83S) addr is the address of the assigned buffer of the first byte of block u. If the block occupying that buffer is not block u and has been UPDATEed it is transferred to mass storage before assigning the buffer. If block u is not already in memory, it is transferred from mass storage into an assigned block buffer. A block may not be assigned to more than one buffer. If u is not an available block number, an error condition exists. Only data within the last buffer referenced by BLOCK or BUFFER is valid. The contents of a block buffer must not be changed unless the

BLOCK-READ

change may be transferred to mass storage. (*F83*) Leaves the address of a buffer containing the given block. Reads the disk if necessary.

Implementations:

MVP: 8088/86:

```
: BLOCK
  'BLOCK @ EXECUTE ;
```

Fig: 8088/86:

```
HEX
: BLOCK
  @ + >R PREV @ DUP @ R - DUP +
  IF BEGIN +BUF 0=
    IF DROP R BUFFER DUP R 1 R/W 2 -
    ENDIF DUP @ R - DUP + 0=
    UNTIL DUP PREV !
  ENDIF R> DROP 2+ ;
```

F83: 8088/86:

```
: BLOCK FILE @ (BLOCK) ;
: (BLOCK)
  (BUFFER) >UPDATE @ 0>
  IF 1 BUFFER# DUP READ-BLOCK 6 + OFF THEN ;
: BLOCK
  FILE @ (BLOCK) ;
```

F-PC:

Not used.

Example:

```
45 BLOCK
```

Entering the desired block number followed by the ideogram leaves the address of the beginning of that block on the top of the stack. If it is not already in a buffer, the block will be read in from disk.

Comment: A useful ideogram which speeds access to disk information when it is not modified. MVP-FORTH has fixed a problem in some earlier implementations of fig-Forth, which sometimes failed to update the last two lines of a source screen on a disk. In MVP-FORTH, this ideogram is vectored to increase flexibility; it defaults to <BLOCK>.

BLOCK-READ BLOCK-WRITE

NOT USED IN MVP-FORTH

(*Fig*) These are the preferred names for the installation dependent code

to read and write one block to the disk. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

These ideograms are obsolete and therefore not implemented.

Comment: Some version of these ideograms may be present to assist in disk I-O. The actual code falls outside the standard and is implementation and hardware dependent.

BMOVE addr1 addr2 n --- MVP-FORTH UTILITY

Move n bytes beginning at address addr1 to addr2. Perform the operation correctly even if the ranges involved overlap. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Pronounced: b-move

Implementations:

MVP: 8088/86:

```
: BMOVE
  ROT ROT DDUP U<
  IF ROT <CMOVE
  ELSE ROT CMOVE THEN ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
HERE PAD 20 BMOVE
```

Will move the twenty bytes beginning at HERE to PAD.

Comment: Using CMOVE with overlapping source and destination fields may have an annoying or disastrous result. Although <CMOVE may be substituted for an offending CMOVE, the BMOVE chooses the correct move order automatically. This becomes one less worry for the programmer.

BODY>

NOT USED IN MVP-FORTH

BOOT

addr1 --- addr2

(Fig) Not defined. (79S) Not defined. (83S *Definition Field Address Conversion Operators*) addr2 is the compilation address corresponding to the parameter field address addr1. (F83) Go from body to code field. (F-PC) Go from body address to code field address cfa.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

: BODY> 2- ;

F-PC:

: BODY> 3 - ;

Example:

```
' BELL DUP >BODY DUP BODY> .S
```

The contrived example will find the code field address of BELL, make a copy and get the body address. The duplicate that and go back to the code field address. Finally print the values on the stack.

Comment: The function is a special set of operators in the 83S. It is not part of the required word set.

BOOT

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) The very first high level word executed during cold start. (F-PC) A deferred word that performs initialization before executing QUIT. You can change this deferred word to perform your own initialization.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
DEFER BOOT ' START IS BOOT
```

F-PC:

```
DEFER BOOT      ' HELLO IS BOOT
```

Example:

```
BOOT
```

The example will simply restart the system.

Comment: The function is deferred until START is defined. In a turnkey program, the run time routine could be substituted.

BOUNDS

NOT USED IN MVP-FORTH

```
addr len --- lim addr
```

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Given address and length, make it ok for DO ... LOOP.. (F-PC) Given address a1 and length n1, return a2 and a3 the boundry addresses for DO ... LOOP.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE BOUNDS

```
DX POP  AX POP  AX DX ADD  2PUSH
```

F-PC:

Same as F83.

Example:

```
: TEST  1 10  BOUNDS  DO  I  U.  LOOP ;
```

TEST will convert the beginning number and the count to the proper bounds for the loop to then print the values 1 through 10.

Comment: The ideogram provides a way to set the parameter for a DO ... LOOP structure from beginning value and a count.

BPDRV

--- n

MVP-FORTH DISK I-O

Find the value in the array BLK/DRV according to the value of DENSITY.

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

BRANCH

Implementations:

MVP: 8088/86:

: BPDRV DENSITY @ 6 MIN 2* BLK/DRV + @ ;

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

BPDRV U.

Get the number of blocks on a drive according to the present value of DENSITY and print it.

Comment: A factored Forth utility used in setting up the system for disk access.

BRANCH

MVP-FORTH

The run-time procedure to unconditionally branch. An in-line offset is added to the interpretive pointer IP to branch ahead or back. BRANCH is compiled by ELSE, AGAIN, REPEAT. (*Fig*) Same as MVP. (79S) Not defined. (83S *System Extension Word Set*) When used in the form: COMPILE BRANCH an unconditional branch operation is compiled. A branch address must be compiled immediately following this compilation address. The branch address is typically generated by following BRANCH with <RESOLVE or >MARK. (F83) Performs an unconditional branch. Notice that we are using absolute addresses instead of relative ones. (fast) (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

CODE BRANCH

HERE LABEL BRAN1 [SI] SI, ADD NEXT JMP

END-CODE

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

CODE BRANCH

```
HERE LABEL BRAN1 0 [IP] IP MOV NEXT
```

F-PC:

```
CODE BRANCH
MOV ES: IP, 0 [IP]
NEXT END-CODE
```

Example:

```
: AGAIN 1 ?PAIRS
  COMPILE BRANCH HERE - , ;
```

The definition of the implementation of AGAIN illustrates the use of this ideogram. "COMPILE BRANCH" generates the pseudo-opcode to which the clause "HERE - , " supplies the displacement.

Comment: A primitive which is not available in some implementations of Forth. The compilation address of BRANCH functions as an unconditional branching opcode for the address interpreter. An in-line branch displacement must follow any compiled instance of this ideogram. These displacements are automatically generated by the control structures. User defined control structures such as CASE may be implemented by using BRANCH and OBRANCH within new immediate compiling ideograms.

CAUTION: Executing BRANCH directly from the terminal or screen will crash your system.

BS --- 8 NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Return the value 8, an ASCII Back Space. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

8 CONSTANT BS

F-PC:

Same as F83.

Example:

```
: DELETE BS EMIT BL EMIT BS EMIT ;
```

BUFFER

DELETE will back space over a character, emit a space and back space over that thus erasing the preceding character.

Comment: Most implementations already have this function defined for the delete key.

BUFFER

n --- addr

MVP-FORTH

Obtain the next block buffer, assigning it to block n. The block is not read from mass storage. If the previous contents of the buffer has been marked as UPDATED, it is written to mass storage. If correct writing to mass storage is not possible, an error condition exists. The address left is the first byte within the buffer for data storage. n is an unsigned number. (*Fig*) Obtain the next memory buffer, assigning it to block n. If the contents of the buffer is marked as updated, it is written to the disc. The block is not read from the disc. The address left is the first cell within the buffer for data storage. (79S) Same as MVP. (83S) Assigns a block buffer to block u. addr is the address of the first byte of the block within its buffer. This function is fully specified by the definition for BLOCK except that if the block is not already in memory it might not be transferred from mass storage. The contents of the block buffer assigned to block u by BUFFER are unspecified. (*F83*) Assigns a buffer to the specified block. No disk read is performed. Leaves the buffer address. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: BUFFER
  USE @ PREV @ =
  IF USE @ +BUF DROP USE ! THEN
  USE @ DUP >R
  BEGIN +BUF UNTIL
  USE ! R@ @ 0< @R
    IF R@ 2+ R@ @ 7FFF AND 0 R/W THEN
    R@ ! R@ PREV ! R> 2+ ;
```

Fig: 8088/86:

```
HEX
: BUFFER  USE @ DUP >R +BUF
  IF USE ! R @ 0<
    IF R 2+ R@ 7FFF AND 0 R/W
  ENDIF ENDIF R ! R PREV ! R> 2 + ;
```

F83: 8088/86:


```
: (BUFFER)    PAUSE  ABSENT?
  IF  ABSENT  1 BUFFER#    4 + @  THEN  ;
: BUFFER      FILE @ (BUFFER)  ;
```

F-PC:

Not used.

Note: The buffers rotate in sequence if there are three or more. Though no problem is apparent to the user, the commented portion is necessary for only two buffers to rotate in sequence. A modified definition would be necessary for use of only one buffer.

Example:

```
: SAVE-BUFFERS  #BUFF  1+  0
  DO  0  BUFFER  DROP  LOOP  ;
```

The implementation of SAVE-BUFFERS illustrates this ideogram and is fully described under its discussion.

Comment: Allows one to enter data into a new block for later storage without reading the current content of that block from the disk. If the buffer is marked for update it will be later written to disk.

BYE

MVP-FORTH

Leave Forth and return to the underlying operating system. (*Fig*) Not defined. (*Fig - 8086/88 only*) Exits Forth and returns to CP/M. (Change back to your CP/M boot disk before typing this word.) (79S) Not defined. (83S) Not defined. (F83) Returns control to ... [DOS]. First it moves the heads down next to the code such that the system is contiguous when saved. Calculates the size in pages. (F-PC) Returns control to DOS. Performs the deferred word BYEFUNC before actually leaving.

Implementations:

MVP: 8088/86:

```
: BYE    FREEZE  0 0 SYSCALL  ;
```

Fig: 8088/86:

```
CODE BYE    EXIT JMP
```

F83: 8088/86:

```
: BYE    0 ROWS @ BIOS-AT  [ ' ] DOS-TYPE IS TYPE  CR
  0 0 BDOS  ;
```

F-PC:

```
: BYE
```

C!

```
RESTORE_VECTORS  BYEFUNC ." Leaving.." 0 0 BDOS ;
```

Example:

```
BYE
```

Entering this ideogram returns to the underlying operating system.

Comment: Though not included in the usual Forth vocabulary it is available in many implementations. It allows one to exit Forth to an underlying operating system.

C! MVP-FORTH

n addr ---

Store the least significant 8-bits of n at addr. (*Fig*) Store 8 bits at address. On word addressing computers, further specification is necessary regarding byte addressing. (*79S*) Same as MVP. (*83S*) The least-significant 8 bits of 16b are stored into the byte at addr. (*F83*) Store an 8-bit value at addr. (*F-PC*) Same as F83.

Pronounced: c-store

Implementations:

MVP: 8088/86:

```
CODE C!  
  BX POP    AX POP    AL [BX], MOV    NEXT JMP    END-CODE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE C!  
  BX POP    AX POP    AL 0 [BX] MOV    NEXT
```

F-PC:

Same as F83.

Example: CAUTION: May corrupt or crash your system.

```
HEX 41 4AC2 C! DECIMAL
```

Place the ASCII value of the character A on the stack followed by an address. This ideogram will store that value in the addressed byte.

Comment: Note that the most significant bits of n are ignored and lost. Since C! will store anywhere in your memory space, take care not to corrupt your dictionary, Forth nucleus, or operating system.

C!L n seg addr --- MVP-FORTH

In an 8086/8088 system, store the least significant 8-bits of n (a byte) in segment, seg, at address addr. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Store the byte at the specified segment and offset.

Implementations:

MVP: 8088/86:

```
CODE C!L
  BX POP    DS POP    AX POP    AL [BX], MOV
  CS BX, MOV  BX DS, MOV  NEXT JMP  END-CODE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

```
CODE C!L
  POP BX    POP DS    POP AX
  MOV 0 [BX], AL  MOV BX, CS  MOV DS, BX
  NEXT      END-CODE
```

Example:

```
HEX
0 40 17 C!L
DECIMAL
```

Set the "Caps Lock" key's flag to 0 for lower case.

Comment: Note that the 'C' implies a character whereas it is a byte which is stored. The 'L' connotes long, that is, out of the current data segment.

C, n --- MVP-FORTH

Store the low order 8 bits of n at the next byte in the dictionary, advancing the dictionary pointer. (*Fig*) Store 8 bits of b into the next available dictionary byte, advancing the dictionary pointer. This is only available on byte addressing computers, and should be used with caution on byte addressing minicomputers. (*79S Reference Word Set*) Same as MVP. (*83S*) Not defined. (*F83*) Same as , except uses an 8-bit value. (*F-PC*) Same as F83.

Pronounced: c-comma

C/L

Implementations:

MVP: 8088/86:

```
: C,   HERE  C!  1  ALLOT  ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Same as MVP.

F-PC:

```
CODE C,  
  MOV BX, UP    MOV AX, DP [BX]  
  INC DP [BX] WORD  MOV BX, AX  
  POP CX    MOV 0 [BX], CL  
  NEXT      END-CODE
```

Example:

```
00  C,
```

Insert the code for an 8080/Z80 NOOP on the top of the dictionary.

Comment: A convenient byte operator for filling dictionary space one byte at a time, as when constructing name headers, machine code sequences, or parameter fields. It may not be available on some 16-bit machines.

C/L

--- n

MVP-FORTH

Constant leaving the number of characters per line; used by the editor.
(Fig) 8080 - Same as MVP. (79S) Not defined. (83S) Not defined.
(F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
HEX  40 CONSTANT C/L
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

C/L .

This ideogram places the value of the length of a line on a screen, and, in the example, the value is printed.

Comment: C/L is used in calculations within the EDITOR. It is not present in all systems, and its value, normally 64, may be terminal dependent.

C@ addr --- byte MVP-FORTH

Leave on the stack the contents of the byte at addr (with higher bits zero, in a 16-bit field). (*Fig*) Leave the 8-bit contents of memory address. On word addressing computers, further specification is needed regarding byte addressing. (*79S*) Same as MVP. (*83S*) 8b is the contents of the byte at addr. (*F83*) Fetch an 8-bit value from addr. (*F-PC*) Same as F83.

Pronounced: c-fetch

Implementations:

MVP: 8088/86:

```
CODE C@   BX BOP   [BX] AL, MOV
          AH AH, SUB   APUSH JMP   END-CODE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE C@
          BX POP    AX AX SUB    0 [BX] AL MOV    1PUSH
```

F-PC:

Same as F83.

Example:

```
HEX 4AC2 C@  EMIT  DECIMAL
```

Place an address on the stack and fetch the byte value at that address. Presuming that it is an ASCII character, then print it.

Comment: Allows fetching individual bytes in memory for inspection or output as in text processing.

CAPS

C@L seg addr --- b MVP-FORTH

In an 8086/8088 system, leave on the stack the contents of the byte in segment seg, at address addr, (with higher bits zero, in a 16-bit field). (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) - Char load long from seg and addr.

Implementations:

MVP: 8088/86:

```
CODE C@L
  BX POP    DS POP    [BX] AL, MOV
  AH AH, SUB  CS BX, MOV  BX DS, MOV
  APUSH JMP   END-CODE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

```
CODE C@L
  POP BX    POP DS    MOV AL, 0 [BX]
  XOR AH, AH  MOV BX, CS  MOV DS, BX
  1PUSH      END-CODE
```

Example:

```
HEX
40 17 C@L .
DECIMAL
```

Check the "Caps Lock" key's flag on the IBM PC or clone, and print it. A zero value is lower case and a value of 40 is upper case.

Comment: This ideogram allows one to examine bytes in memory which are outside of the segments used by Forth. Though the 'C' connotes character, one is actually fetching the byte. The 'L' connotes long, that is, outside of the Forth segments. It is used in the *UTILITY* for a "dump long", *DUMPL*.

CAPS --- addr NOT USED IN MVP-FORTH

(*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) If true, then convert names to upper case. (*F-PC*) Same as *F83*.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

VARIABLE CAPS

F-PC:

Same as F83.

Example:

```
: COMPARE    CAPS @
  IF CAPS-COMP ELSE  COMP  THEN;
```

The example is taken from the F-PC kernel source code.

Comment: A number of functions need to be modified to make the implementation case insensitive. This is one of them.

CASE --- addr n NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined (F83) Not defined. (F-PC) CASE (--)

Start a CASE statement, as follows:

```
value1
CASE  value2 OF func2      ENDOF
      value3 OF func3      ENDOF
      otherwise_func
      ( DROP )
ENDCASE
```

Not needed as of 05/17/89

Now does the DROP itself.

Allows an arbitrary value to specify a function be performed. Value1 is compared against value2 and value3. The appropriate func2, func3, or otherwise_func is performed. If your values are sequential, the EXEC: word is much faster, and more space efficient. See also EXEC:

(Eaker) Used in a colon definition in the form: CASE ... OF ... ENDOF ... ENDCASE. Note that OF ... ENDOF pairs may be repeated as necessary. At compile-time CASE saves the current value of CSP and resets it to the current position of the stack. This information is used by ENDCASE to resolve forward references left on the stack by any ENDOF's which precede it. n is left for subsequent error checking. CASE has no run-time effects.

CFA

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

```
: CASE
  COMPILE DOCASE    <MARK 0 ; IMMEDIATE
```

Eaker:

```
: CASE
  ?COMP CSP @ !CSP 4 ; IMMEDIATE
```

Example:

```
: GEN CASE
  0 OF IMMEDIATE ENDOF
  10 OF DIRECT ENDOF
  20 OF INDEXED ENDOF
  30 OF EXTENDED ENDOF
  MODE-ERROR
  ENDCASE RESET ;
```

The example is taken from the original article by Charles E. Eaker. The MODE-ERROR and RESET functions are taken from his implementation of fig-Forth and are not in common usage. They could be omitted from the example.

Comment: This is probably the most commonly used CASE function. It was published in *Forth Dimensions* Vol. II, No. 3, 1980 as one of three winners of FIG's CASE Statement Contest. All entries to the contest were published in that issue.

CFA

pfa --- cfa

MVP-FORTH

Convert the parameter field address of a definition to its code field address. (*Fig*) Same as MVP. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: CFA 2- ;
```


Fig: 8088/86:

: CFA 2 - ;

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
' ?CONFIGURE CFA EXECUTE
```

Find the parameter field address of the ideogram ?CONFIGURE and convert it to the code field address so that it can be executed.

Comment: A function which is needed in an indirect threaded code implementation of Forth, to allow execution of ideograms after being found in the dictionary. This practice, useful in some applications, is implementation dependent and violates the usage constraints of the 79S.

CHANGE

MVP-FORTH

Modify the size of your Forth image and the number of buffers in use according to the current values of LIMIT and #BUFF. (*Fig*) Not defined. (79S) Not defined. (83S) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations: Note: The following auxiliary definitions are compiled headerless from source code. One can add those ideograms to his vocabulary.

MVP: 8088/86:

```
: CHANGE FREEZE LIMIT HDBT #BUFF * -
  DUP ' FIRST ! US - DUP RTS -
  DUP INIT-USER !
  [ INIT-USER 4 + ] LITERAL !
  DUP [ INIT-USER 2+ ] LITERAL !
  OVER RPP ORIGIN HERE !
  HERE ROT ROT ! ROT ROT ! EXECUTE ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

CLEAR

Not used.

Note: HDBT is compiled to a hex literal value 404 which is the number of bytes in a block buffer plus four. US is compiled to a hex literal value 52 which is the size of the user area. RTS is compiled to a hex literal value of A0 which is the offset below the return stack pointer at which the stack pointer is located.

Example:

```
HEX
8000 ' LIMIT !
4 ' #BUFF !
CHANGE
DECIMAL
```

First set the constants LIMIT and #BUFF to the desired values, then execute CHANGE.

Comment: This ideogram lets you dynamically alter the number and location of your block buffers. Simply modify the values in the constants LIMIT and #BUFF and enter CHANGE. All the other buffer management parameters, including the startup data in low memory, are adjusted accordingly. This allows much greater flexibility in deployment of your available RAM.

CLEAR

n ---

MVP-FORTH

Clear Screen n to all blanks. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: CLEAR OFFSET @ + BUFFER 400
  BL FILL UPDATE ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: WIPE SCR @ CLEAR ;
```

This ideogram could be used to define WIPE for an EDITOR.

Comment: This ideogram performs the function in the original fig-Forth EDITOR. It is included in the MVP-FORTH implementation so that rudimentary screen editing can be done with the ideogram PP .

CMOVE

addr1 addr2 n ---

MVP-FORTH

Move n beginning at address addr1 to addr2. The contents of addr1 is moved first proceeding toward high memory. If n is zero or negative nothing is moved. (*Fig*) Move the specified quantity of bytes beginning at address from to address to. The contents of address from is moved first proceeding toward high memory. Further specification is necessary on word addressing computers. (79S) Same as MVP. (83S) Move u bytes beginning at address addr1 to addr2. The byte at addr1 is moved first, proceeding toward high memory. If u is zero nothing is moved. (F83) Move a set of bytes from the from address to the to address. The number of bytes to be moved is count. The bytes are moved from low address to high address, so overlap is possible and in fact sometimes desired. (F-PC) Same as F83.

Pronounced: c-move

Implementations:

MVP: 8088/86:

```
: CMOVE  DUP  1  <
  IF  DDROP  DROP  ELSE  <CMOVE>  THEN  ;
```

Fig: 8088/86:

```
CODE CMOVE
  CLD  SI BX, MOV  CX POP  DI POP  SI POP
  DS AX, MOV  AX EX, MOV  REP  AL AL, MOVS
  BS SI, MOV  NEXT JMP
```

F83: 8088/86:

```
CODE CMOVE
  CLD  IP BX MOV  DS AX MOV  AX ES MOV
  CX POP  DI POP  IP POP
  REP  BYTE MOVS  BX IP MOV  NEXT
```

F-PC:

```
CODE CMOVE
  MOV BX, IP  MOV AX, DS  POP CX
  POP DI  POP IP  MOV DX, ES  MOV ES, AX
  REP NZ  MOVS  MOV IP, BX  MOV EX, DX
  NEXT  END-CODE
```

Example:

```
HEX  4AC2  6AC2  20  CMOVE  DECIMAL
```

Place the source address and then the destination address followed by a count on the stack followed by the ideogram and the move of 20 bytes will be accomplished. Make sure that it won't write over anything important.

Comment: This ideogram will propagate data toward high memory when source and destination fields overlap. Although such an effect may sometimes be desirable, BMOVE averts that phenomenon.

CMOVE>

NOT USED IN MVP-FORTH

```
addr1 addr2 u ---
```

(Fig) Not defined. (79S) Not defined. (83S) Move the u bytes at address addr1 to addr2. The move begins by moving the byte at (addr1 plus u minus 1) to (addr2 plus u minus 1) and proceeds to successively lower addresses for u bytes. If u is zero nothing is moved. (Useful for sliding a string towards higher addresses). (F83) Same as CMOVE above except that bytes are moved in the opposite direction, i.e., from high addresses to low addresses. (F-PC) Same as f83.

Implementations:

MVP: 8088/86:

Fig: 8088/86:

F83: 8088/86:

```
CODE CMOVE>
```

```
STD  IP BX MOV  DS AX MOV  AX ES MOV  CX POP
CX DEC  DI POP  IP POP  CX DI ADD  CX IP ADD
CX INC
REP  BYTE MOVS  BX IP MOV  CLD  NEXT
```

F-PC:

```
CODE CMOVE>
```

```
MOV BX, IP  MOV AX, DS  POP CX  DEC CX  POP DI
POP IP
ADD DI, CX  ADD IP, CX  INC CX  MOV DX, ES  MOV
ES, AX
STD  REPNZ  MOVSB  CLD  MOV IP, BX  MOV EX, DX
NEXT  END-CODE
```

Example:

```
HEX  4AC3  4AC4  20  CMOVE>  DECIMAL
```

With the normal CMOVE, the data from 4AC4 up would be written over with the value in 4AC3 and the information would be lost. Instead, CMOVE> shifts 32 bytes one position to the right.

Comment: This ideogram works exactly like CMOVE, except when its source and destination fields overlap. The ideogram has the same function as <CMOVE in earlier Forth implementations. The ideogram BMOVE, a better solution, intelligently chooses the action so that overlapping fields hold no surprises for the programmer.

CODE | <name> --- NOT USED IN MVP-FORTH

CODE is part of which ever assembler is added to MVP-FORTH. (Fig) Not defined. (79S) Not defined. (83S Assembler Extension Word Set) A defining word executed in the form: CODE <name> ... END-CODE. Creates a dictionary entry for <name> to be defined by a following sequence of assembly language words. Words thus defined are called code definitions. This newly created word definition for <name> cannot be found in the dictionary until the corresponding END-CODE is successfully processed (see: END-CODE). Executes ASSEMBLER . sys is balanced with its corresponding END-CODE. (F83) CODE is the defining word for Forth assembler definitions. It saves the context vocabulary and hides the name. (F-PC) Define "name" as a new code definition. Assembly language follows, terminated by END-CODE.

Implementations:

MVP: 8088/86:

```
: CODE
  CREATE SMUDGE HERE DUP 2- ! ENTERCODE ;
: ENTERCODE [COMPILE] ASSEMBLER SP@ ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
Assembler implementation: -
: CODE
  [ ASSEMBLER ] CONTEXT @ AVOC !
  LABEL HIDE HERE DUP 2- ;
```

Note: Versions of F83 differ.

F-PC:

COLD

```
: CODE LABEL -3 DP +! HIDE ;
```

Example:

```
CODE NOOP NEXT END-CODE
```

The example is from F-PC. The function introduces a time delay but does nothing.

Comment: The actual implementation of CODE is implementation dependent. It is available in most Forth systems.

COLD

MVP-FORTH

The cold start procedure to adjust the dictionary pointer to the minimum standard and restart via ABORT . May be called from the terminal to remove application programs and restart. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) The high level cold start code. For ordinary Forth, BOOT should initialize and pass control to QUIT. (F-PC) The high-level cold-start code. For ordinary Forth, BOOT should initialize and pass control to QUIT. Typically Boot contains HELLO.

Implementations:

MVP: 8088/86:

```
: COLD EMPTY-BUFFERS
  INIT-USER UP @ 6 + 30 CMOVE
  PAGE ." MVP-FORTH VERSION 1.xx.03 " CR
  1 DENSITY ! FIRST USE ! FIRST PREV !
  DR0 0 EPRINT !
  INIT-FORTH @ ' FORTH 2+ !
  DECIMAL ABORT ;
```

Fig: 8088/86:

HEX

```
: COLD
  EMPTY-BUFFERS 0 DENSITY ! FIRST USE !
  FIRST PREV ! DR0 0 EPRINT !
  ORIGIN 12 + UP @ 6 + 10 CMOVE
  ORIGIN 0C + @ FORTH 6 + ! ABORT ;
```

F83: 8088/86:

```
: COLD
  INIT BOOT QUIT ;
```

F-PC:

```
: COLD
```

```
[ LABEL COLDBODY ] 1STCOLD
IF SEGSET VMODE.SET INITSTUFF THEN
%OFF> 1STCOLD BOOT QUIT ;
```

Example: CAUTION — This will erase all new definitions.

COLD

This ideogram will cause the version of Forth in memory to be reconfigured to its condition on start up. All source code and application programs loaded upon the start up image are lost.

Comment: This ideogram is a start-up primitive present only in some versions of Forth. In other versions other ideograms may be used to remove application programs from the dictionary. Some versions of Forth overwrite the startup information and it is impossible to restart without rebooting the entire system. COLD is similar to EMPTY.

COMPARE

NOT USED IN MVP-FORTH

a1 a2 n1 --- f1

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Performs a string compare. If CAPS is true, characters from both strings are converted to upper case before comparing. (F-PC) Perform a string compare. If CAPS is true, characters from both strings are converted to upper case before comparing. The strings are not changed in memory.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

: COMPARE

CAPS @ IF CAPS-COMP ELSE COMP THEN ;

F-PC:

Same as F83.

Example:

PAD PAD 100 + 100 COMPARE

Compare 100 bytes starting at PAD with those starting at PAD plus 100 and return a zero flag if they are the same.

Comment: This ideogram or some variation of it can be used in many

CONFIGURE

sort routines.

COMPILE

MVP-FORTH

When a word containing COMPILE executes, the 16-bit value following the compilation address of COMPILE is copied (compiled) into the dictionary, that is COMPILE DUP will copy the compilation address of DUP. (*Fig*) When the word containing COMPILE executes, the execution address of the word following COMPILE is copied (compiled) into the dictionary. This allows specific compilation situations to be handled in addition to simply compiling an execution address (which the interpreter already does). (79S) Same as MVP. (83S) When <name> is executed, the compilation address compiled for <namex> is compiled and not executed. <name> is typically immediate and <namex> is typically not immediate. See: "compilation" (F83) Compile the following word when this def[inition] executes. (F-PC) Compile the (typically not-immediate) following word <name> when this definition executes. Name is later compiled into the LIST dictionary space.

Form: COMPILE [0 ,] (will copy a zero).

Implementations:

MVP: 8088/86:

```
: COMPILE ?COMP R> DUP 2+ >R @ , ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
: COMPILE >R DUP 2+ R> @ , ;
```

F-PC:

```
: COMPILE 2R@ R> 2+ >R @L X, ;
```

Example:

```
: LITERAL STATE @  
  IF COMPILE LIT , THEN ;  
  IMMEDIATE
```

The implementation of LITERAL provides an example of this ideogram.

Comment: COMPILE is usually used within immediate words such as IF and DOES>.

CONFIGURE

MVP-FORTH

This ideogram lets you change the number of drives available on your

system and the physical sector formatting used on each drive. First, it asks you to key in the total number of drives — a digit from 1 through 5; any other key will abort the process, changing nothing. Once the number of drives is set, you are prompted to enter a “density code” for each drive starting with drive 0.

0 - 5"	Single Sided, Single Density	(5-SSSD)
1 - 8"	Single Sided, Single Density	(8-SSSD)
2 - 8"	Double Sided, Single Density	(8-DSSD)
3 - 8"	Single Sided, Double Density	(8-SSDD)
4 - 8"	Double Sided, Double Density	(8-DSDD)
5 - 8"	Single Sided, Extended Density	(8-SSEXT)
6 - 8"	Double Sided, Extended Density	(8-DSEXT)

No density parameters will be updated until a correct code has been keyed in for every requested drive. Thus, this stage of CONFIGURE may be nondestructively aborted by the user at any time. On successful completion, OFFSET is cleared, directing BLOCK access to drive 0. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: CONFIGURE    ?CONFIGURE
CR ." NUMBER OF DRIVES ? " KEY 31
- DUP 5 U< NOT
ABORT" TOO MANY DRIVES"
DUP 31 + EMIT 1+ ' MAX-DRV !
MAX-DRV 0
DO CR ." DRIVE " I . ." ? "
KEY 30 - DUP 7 U< NOT
ABORT" OUT OF RANGE"
DUP 30 + EMIT I 2* DEN + !
LOOP
DR0 CR CR ." DR0 SELECTED " CR ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

CONSTANT

Example:

CONFIGURE

The system shows you the current drive settings, and then prompts you to change them. The density codes refer to the order of items in the arrays SEC/TR and BLK/DRV. They can be easily examined with DUMP and modified as necessary.

Comment: This ideogram is designed to be used from the terminal. For program control of drive configuration the array DEN and the alterable constant MAX-DRV may be modified directly. The use of dynamically adjustable disk parameters gives greater flexibility to the MVP-FORTH system. Note that DR0 is always reselected on completion, since the previous value of OFFSET may no longer be correct. The new disk configuration parameters may be permanently stored into the cold-start system image on disk. (See FREEZE)

CONSTANT

n ---

MVP-FORTH

A defining word to create a dictionary entry for <name>, leaving n in its parameter field. When <name> is later executed, n will be left on the stack. (*Fig*) A defining word used ... to create word cccc, with its parameter field containing n. When cccc is later executed, it will push the value of n to the stack. (79S) Same as MVP. (83S) A defining word executed in the form: 16b CONSTANT <name>. Creates a dictionary entry for <name> so that when <name> is later executed, 16b will be left on the stack. (F83) A defining word that creates constants. At runtime the value of the constant is placed on the stack. (F-PC) Same as F83.

Form: n CONSTANT <name>

Implementations:

MVP: 8088/86:

```
: CONSTANT CREATE ,  
  ;CODE DX INC DX BX, MOV [BX] AX, MOV  
  APUSH JMP END-CODE
```

Fig: 8088/86:

```
: CONSTANT CREATE SMUDGE , ;CODE  
  HERE LABEL DOCON DX INC DX BX, MOV  
  [BX] AX, MOV APUSH JMP
```

F83: 8088/86:

```
: CONSTANT CREATE , ;USES DOCONSTANT ,
```

F-PC:

```
: CONSTANT
  HEADER , JUMP , :USES DOCONSTANT , -X
```

Note: ;CODE is defined in the cross compiler's ASSEMBLER and not in the MVP- FORTH nucleus.

Example:

```
HEX 20 CONSTANT BL DECIMAL
```

This example is the implementation of BL, the ASCII value of the space character in hex.

Comment: The use of CONSTANT is more efficient than VARIABLE if the value is not to be changed frequently.

CONTEXT

--- addr

MVP-FORTH

Leave the address of a variable specifying the vocabulary in which dictionary searches are to be made, during interpretation of the input stream. (*Fig*) A user variable containing a pointer to the vocabulary within which dictionary searches will first begin. (*79S*) Same as MVP. (*83S System Extension Word Set*) The address of a variable which determines the dictionary search order. (*F83*) The array specifying the search order. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

```
HEX 3C USER CONTEXT
```

Fig: 8088/86:

```
HEX 20 USER CONTEXT
```

F83: 8088/86:

```
: CONTEXT CURRENT 2+ ;
```

F-PC:

```
VARIABLE CONTEXT
```

Example:

```
: DEFINITIONS CONTEXT @ CURRENT ! ;
```

The implementation of DEFINITIONS illustrates how this ideogram is used.

Comment: Some implementations of will chain multiple vocabularies for searching while 79S requires every vocabulary to chain to the Forth vocabulary. CONTEXT contains a pointer to a pointer to the length byte

CONVERT

of the first name to be searched, but this may vary among implementations.

CONVERT d1 addr1 --- d2 addr2 MVP-FORTH

Convert to the equivalent stack number the text beginning at addr1+1 with regard to BASE. The new value is accumulated into double number d1, being left as d2. addr2 is the address of the first non-convertable character. (*Fig*) Not defined. (*79S*) Same as MVP. (*83S*) +d2 is the result of converting the characters within the text beginning at addr1+1 into digits, using the value of BASE, and accumulating each into +d1 after multiplying +d1 by the value of BASE. Conversion continues until an unconvertible character is encountered. addr2 is the location of the first unconvertible character. (*F83*) Starting with the unsigned double number ud1 and the string at adr1, convert the string to a number in the current base. Leave result and address of unconvertable digit on [the] stack. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

```
: CONVERT
  BEGIN 1+ DUP >R C@ BASE @ DIGIT WHILE
    SWAP BASE @ U* DROP ROT BASE @ U*
    D+ DPL @ 1+
    IF 1 DPL +! THEN
      R
  REPEAT R> ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: CONVERT
  BEGIN 1+ DUP >R C@ BASE @ DIGIT
  WHILE SWAP BASE @ UM* DROP ROT
    BASE @ UM* D+ DOUBLE?
    IF 1 DPL +! THEN R>
  REPEAT DROP R> ;
```

F-PC:

```
: CONVERT
  BEGIN 1+ DUP>R C@ BASE @ DIGIT
  WHILE SWAP BASE @ UM* DROP ROT
    BASE @ UM* D+ DOUBLE?
```

```
IF DPL INCR THEN R>
REPEAT DROP R> ;
```

Example:

```
HEX 0 0 4AC2 CONVERT DECIMAL
```

Place a double precision value of 0 on the stack and then an address in memory at which the ideogram is to begin its conversion.

Comment: This ideogram replaces the now obsolete (NUMBER) in the older fig-Forth. As you might expect, no error message is given if the numeric text converts to a number larger than 32 bits. Any higher bits are lost.

COPY MVP-FORTH UTILITY

n1 n2 ---

Copy the contents of screen n1 to screen n2. (*Fig*) Editor - Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) - Perform a DOS COPY with the filespec following the COPY command.

Implementations:

MVP: 8088/86:

```
: COPY OFFSET @ + SWAP BLOCK 2- ! UPDATE ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: COPY 2 ?ENOUGH FLUSH (COPY) FLUSH ;
```

F-PC:

```
: COPY ( <filespec> --- )
" COPY " 0 "SYSCOMMAND;
```

Example:

```
20 120 COPY
```

Copies screen 20 to screen 120 .

Comment: This ideogram is defined in the vocabulary of most editors. However, it is convenient to have it defined in the Forth vocabulary, too.

COUNT MVP-FORTH

addr --- addr+1 n

Leave the address addr+1 and the character count of text beginning

at addr. The first byte at addr must contain the character count n. Range of n is 0..255. (*Fig*) Leave the byte address addr2 and byte count n of a message text beginning at address addr1. It is presumed that the first byte at addr1 contains the text byte count and the actual text starts with the second byte. Typically COUNT is followed by TYPE. (79S) Same as MVP. (83S) addr2 is addr1+1 and +n is the length of the counted string at addr1. The byte at addr1 contains the byte count +n. Range of +n is {0..255}. (F83) Given the address on the stack, returns the address plus one and the byte at that address. Useful for strings. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

```
: COUNT  DUP  1+  SWAP  C@  ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE COUNT
```

```
  BX POP    AX AX SUB    0 [BX] AL MOV    BX INC  
  BX PUSH   1PUSH
```

F-PC:

Same as F83.

Example:

```
PAD  COUNT  TYPE
```

PAD leaves an address of a byte which contains the number of characters presently in PAD. This byte is placed on the stack on top of an address which increased by one.

Comment: A one-byte count limits string length to 255 characters. Although intended primarily for string handling, this ideogram will work for any kind of data.

Cause a carriage-return and line-feed to occur on the current output device. (*Fig*) Transmit a carriage return and line feed to the selected output device. (79S) Same as MVP. (83S) Displays a carriage-return and line-feed or equivalent operation. (F83) Typically set to CRLF, above. (F-PC) Typically set to CRLF, above. PR-STAT Return printer status, if implemented, else TRUE (PRINT) The value of the DEFERRED word EMIT when you want to send a character to the printer.

Implementations:

MVP: 8088/86:

```
: CR 'CR @ EXECUTE ;
```

Fig: 8088/86:

```
CODE CR PCR JMP
```

F83: 8088/86:

```
: CRLF
  13 EMIT 10 EMIT #OUT OFF 1 #LINE +! ;
DEFER CR
' CRLF IS CR
```

F-PC:

```
DEFER CR
: CRLF
  13 EMIT 10 EMIT #OUT OFF
  #LINE DUP @ 1+
  PRINTING @ 0=
  IF ROWS 1- MIN THEN
    SWAP ! ;
' CRLF IS CR
```

Example:

```
CR CR ." HELLO"
```

The two ideograms cause the display to advance two lines and then print HELLO at the left margin of the display.

Comment: Placing the ideogram, CR, in the input stream is not equivalent to pressing the "Return" key on the terminal keyboard. Because of differences among output devices in the inclusion of line-feeds, this ideogram is vectored. This will simplify modification if necessary. Normally, CR vectors to <CR>.

CREATE

MVP-FORTH

A defining word used to create a dictionary entry for <name>, without allocating any parameter field memory. When <name> is subsequently executed, the address of the first byte of <name>'s parameter field is left on the stack. (*Fig*) A defining word used in the form: CREATE cccc, by such words as CODE and CONSTANT to create a dictionary header for a Forth definition. The code field contains the address of the words parameter field. The new word is created in the CURRENT vocabulary. (79S) Same as MVP. (83S) A defining word executed in the form:

CREATE

CREATE <name>. Creates a dictionary entry for <name>. After <name> is created, the next available dictionary location is the first byte of <names>'s parameter field. When <name> is subsequently executed, the address of CREATE does not allocate space in <names>'s parameter field. (F83) Define a word that returns the address of the next available user memory location. (F-PC) Same as F83.

Form: CREATE <name>

Implementations:

MVP: 8088/86:

```
: CREATE  BL WORD DUP DUP 1+ C@ 0 =
  ABORT" ATTEMPTED TO REDEFINE NULL"
  DUP CONTEXT @ @ <FIND>
  IF DDROP WARNING @
    IF DUP COUNT TYPE SPACE
      ."  ISN'T UNIQUE "
    THEN THEN C@ WIDTH @ MIN 1+ ALLOT DUP 80
    TOGGLE HERE 1- 80 TOGGLE LATEST ,
    2 ALLOT CURRENT @ ! ;CODE
  D INX D PUSH NEXT JMP END-CODE
```

Fig: 8088/86:

HEX

```
: CREATE  -FIND
  IF DROP NFA ID. 4 MESSAGE SPACE ENDIF
  HERE DUP C@ WIDTH @ MIN 1+ ALLOT DUP 0A0 TOGGLE
  HERE 1 - 080 TOGGLE LATEST , CURRENT @ ! HERE 2+ , ;
```

F83: 8088/86:

```
: "CREATE  WARNING @
  IF DUP FIND NIP
    IF HERE COUNT TYPE ."  isn't unique " THEN
  THEN DUP COUNT >HEADER 6 +
  PLACE ( lay down name )
  DUP 1+ @ CURRENT @ HASH DUP @ >HEADER ! ( link )
  DP-H @ DUP 6 + LAST ! SWAP ! ( fix thread )
  BLK @ DUP IF VIEW# @ 256 * + THEN
  >HEADER 2+ ! ( view )
  HERE >HEADER 4 + ! ( code field )
  >HEADER 6 + COUNT TUCK + C! C@ 8 +
  >HEADER SWAP ,HEADER
  COMPILE [ [FORTH] ASSEMBLER DCREATE , META ] ;
: CREATE ( -- ) TOKEN "CREATE ;
```


F-PC:

```
: CREATE      HEADER ,CALL ;USES NEXT , -X
```

Example:

```
: VARIABLE    CREATE 2 ALLOT ;
```

The implementation of VARIABLE illustrates how this ideogram is used to create a new header for an ideogram in the dictionary.

Comment: Under 79S, CREATE builds a VARIABLE header with no parameter space. In contrast, fig-Forth's CREATE sets up a CODE definition which will jump into the parameter field and crash unless ;CODE is used to redirect the code address toward a valid machine language routine. 79S's CREATE now replaces the older ideogram in generating defining words. Care must be taken by those already familiar with this word when they move to different implementations of Forth. Note: Since the ASCII null is used as an ideogram to terminate interpretation from terminal and disk buffers, it must not be redefined.

CSP

--- addr

MVP-FORTH

A user variable temporarily storing the stack pointer position, for compilation error checking. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Used for compile time error checking. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

```
HEX
3E  USER  CSP
```

Fig: 8088/86:

```
HEX
2C  USER  CSP
```

F83: 8088/86:

```
VARIABLE CSP
```

F-PC:

Same as F83.

Example:

```
: !CSP  SP@  CSP  !  ;
```

This example comes from fig-Forth and is not used in MVP-FORTH.

Comment: CSP stands for "check stack position". If the stack position

D!

is changed within a colon definition, an error condition is raised. This compiler security is a fig-Forth feature outside the scope of the 79S.

CURRENT

--- addr

MVP-FORTH

Leave the address of a variable specifying the vocabulary into which new word definitions are to be entered. (*Fig*) Omitted from the *Installation Manual* by error. (79S) Same as MVP. (83S *System Extension Word Set*) The address of a variable specifying the vocabulary in which new word definitions are appended. (F83) New words are added to the CURRENT vocabulary. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

HEX

40 USER CURRENT

Fig: 8088/86:

HEX

22 USER CURRENT

F83: 8088/86:

: CURRENT >CONTEXT @ ;

F-PC:

VARIABLE CURRENT

Example:

: LATEST CURRENT @ @ ;

The implementation of LATEST illustrates the use of this variable.

Comment: Early printings of the fig-Forth Glossary left out this ideogram. Now it is present in it and most other implementations of Forth. CURRENT contains a pointer to a pointer to the length byte of the latest name in the current vocabulary, but this may vary among implementations.

D!

d addr ---

MVP-FORTH

Store d as a double precision integer. (*Fig*) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Pronounced: d-store

Implementations:

MVP: 8088/86:

```
CODE D!    BX POP    AX POP
    AX [BX], MOV    AX POP    AX [BX], 2+ MOV
NEXT JMP    END-CODE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example: CAUTION: May corrupt or crash your system.

HEX 33.33 4AC2 D! DECIMAL

Enter the double precision value on the stack taking four bytes. Then remove these four bytes from the stack and store them in four bytes beginning at memory address 4AC2. 79S leaves the order of the bytes in each pair unspecified.

Comment: A useful operation whose function is provided in the extended vocabulary of 79S. The ideogram D! is used in MVP-FORTH to avoid the misleading connotations of the numeral 2. The synonym 2! may be loaded as an alias for *Double Number Word Set* compatibility.

D+ d1 d2 --- d3 MVP-FORTH

Leave the arithmetic sum of d1 plus d2. (*Fig*) Leave the double number sum of two double numbers. (79S) Same as MVP. (83S) wd3 is the arithmetic sum of wd1 plus wd2. (*F83*) Add the two double precision numbers on the stack and return the result as a double precision number. (*F-PC*) Same as F83.

Pronounced: d-plus

Implementations:

MVP: 8088/86:

```
CODE D+    AX POP    DX POP
BX POP     CX POP    CX DX, ADD    BX AX, ADC
DPUSH JMP   END-CODE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE D+
    AX POP    DX POP    BX POP    CX POP
```

D-

CX DX ADD BX AX ADC 2PUSH

F-PC:

Same as F83

Example:

2.2 3.3 D+

Place two double precision values on the stack and add them leaving the sum, 55, as a double precision value on the stack. Note that the location of the decimal point is not maintained.

Comment: A primitive present in most versions of Forth with which other double precision ideograms can be defined.

D+-

d1 n --- d2

MVP-FORTH

Apply the sign of n to the double number d1, leaving it as d2. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Pronounced: d-plus-minus

Implementations:

MVP: 8088/86:

: D+- 0< IF DNEGATE THEN ;

Fig: 8088/86:

: D+- 0< IF DMINUS ENDIF ;

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

2.2 -1 D+-

Place a double precision value on the stack followed by a single precision negative value and then the ideogram to convert, in this case, the 22 to -22. Note that the position of the decimal point is lost.

Comment: Similar to the ideogram, +- , for single precision numbers and useful for double precision number manipulations.

D-

d1 d2 --- d3

MVP-FORTH UTILITY

Subtract d2 from d1 and leave the difference d3. (*Fig*) Not defined.

(79S Extension Word Sets) Same as MVP. (83S Double Number Extension Word Set) wd3 is the result of subtracting wd2 from wd1. (F83) Subtract the two double precision numbers. (F-PC) Subtract double number at top from second double number.

Pronounced: d-minus

Implementations:

MVP: 8088/86:

```
: D-    DNEGATE    D+    ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
CODE D-
```

```
AX POP    DX POP    BX POP    CX POP
CX DX SUB    BX AX SBB    2PUSH
```

F-PC:

Same as MVP.

Example:

```
3.3  2.2  D-
```

Place two double precision values on the stack and subtract them leaving the difference as the double precision value, 11 . Note that the location of the decimal point is not maintained.

Comment: This is part of the 79S Double Number Word Set which can be defined easily if it is not already present.

D. d ---

MVP-FORTH

Display d converted according to BASE in a free-field format, with one trailing blank. Display the sign only if negative. (Fig) Print a signed double number from a 32-bit two's complement value. The high-order 16 bits are most accessible on the stack. Conversion is performed according to the current BASE. A blank follows. (79S Extension Word Sets) Same as MVP. (83S Double Number Extension Word Set) The absolute value of d is displayed in a free field format. A leading negative sign is displayed if d is negative. (F83) Output as a signed double number with a trailing space. (F-PC) Same as F83.

Pronounced: d-dot

D.R

Implementations:

MVP: 8088/86:

```
: D. 0 D.R SPACE ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
: D. (D.) TYPE SPACE ;
```

F-PC:

Same as F83.

Example:

```
2.2 D.
```

Place a double precision value on the stack and then print it. Note that the location of the decimal point is not maintained but the number of significant digits is.

Comment: The counterpart of . with single precision numbers to display double precision numbers.

D.R d n ---

MVP-FORTH

Display d converted according to BASE, right aligned in an n character field. Display the sign only if negative. (*Fig*) Print a signed double number d right aligned in a field n characters wide. (*79S Extension Word Sets*) Same as MVP. (*83S Double Number Extension Word Set*) d is converted using the value of BASE and then displayed right aligned in a field +n characters wide. A leading minus sign is displayed if d is negative. If the number of characters required to display d is greater than +n, an error condition exists. See: "number conversion". (*F83*) Output as a signed double number right justified. (*F-PC*) Same as F83.

Pronounced: d-dot-r

Implementations:

MVP: 8088/86:

```
: D.R DEPTH 3 <
  ABORT" EMPTY STACK"
  >R SWAP OVER DUP D+-
  <# #S ROT SIGN #>
  R> OVER - SPACES TYPE ;
```

Fig: 8088/86:

```
: D.R
```

```
>R  SWAP  OVER  DABS  <#  #S  SIGN  #>
R>  OVER  -  SPACES  TYPE  ;
```

F83: 8088/86:

```
: D.R  >R  (D.)  R>  OVER  -  SPACES  TYPE  ;
```

F-PC:

Same as F83.

Example:

```
2.2  10  D.R
```

Enter a double precision value, 22 , and then a single precision value for the field width, ten. This ideogram then prints the value 22 as two digits preceded by 8 leading blanks. Note that the location of the decimal point is lost but that all of the digits are printed.

Comment: Allows formatting the output of double precision numbers. It is actually the primitive for many of the other ideograms which output both single and double precision numbers in some implementations of Forth including MVP- FORTH. Note that it traps an empty stack before garbage is printed.

D0= d --- flag

MVP-FORTH UTILITY

Leave true if d is zero. (*Fig*) Not defined. (*79S Extension Word Sets*) Same as MVP. (*83S Double Number Extension Word Set*) flag is true if wd is zero. (*F83*) Compare the top double number to zero. True if d = 0. (*F-PC*) Same as F83.

Pronounced: d-zero-equal

Implementations:

MVP: 8088/86:

```
: D0=  OR  0=  ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Same as MVP.

F-PC:

Same as MVP.

Example:

```
2. 2.  D0=
```

D=

Enter a double precision value and test it. In this case a 0 flag will be left.

Comment: A logical test for double precision numbers. The test destroys the comparand.

D< d1 d2 --- flag MVP-FORTH

True if d1 is less than d2. (*Fig*) Not defined. (*79S*) Same as MVP. (*83S*) flag is true if d1 is less than d2 according to the operation of < except extended to 32 bits. (*F83*) Compare the top two double numbers. True if $d1 < d2$. (*F-PC*) Same as F83.

Pronounced: d-less-than

Implementations:

MVP: 8088/86:

```

: D<    ROT    DDUP    =
IF ROT ROT    DNEGATE D+    0<
ELSE SWAP <    SWAP DROP
THEN SWAP DROP ;

```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: D<    2 PICK OVER =
  IF    DU<    ELSE  NIP ROT DROP <  THEN ;
```

F-PC:

Same as F83.

Example:

2.2 3.3 D<

Enter two double precision values on the stack and then compare them. In this case the value of the flag left on the stack is 1. Note that the location of the decimal point is lost.

Comment: Though not present in many versions of Forth, it provides a convenient tool when working with double precision numbers. The comparison destroys both double precision comparands.

```
D=      d1 d2 --- flag      MVP-FORTH UTILITY
```

True if d1 equals d2. (*Fig*) Not defined. (*79S Extension Word Set*)
Same as MVP. (*83S Double Number Extension Word Set*) *flag* is true

if wd1 equals wd2. (*F83*) Not defined. (*F-PC*) Compare the top two double numbers. True if d1 = d2.

Pronounced: d-equal

Implementations:

MVP: 8088/86:

: D= D- D0= ;

Fig: 8088/86:

Not used.

F83: 8088/86:

Same as MVP.

F-PC:

Same as MVP.

Example:

2.2 3.3 D=

Enter two double precision values on the stack and compare them. In this case the flag left on the stack is 0. Note that the location of the decimal point is not considered.

Comment: Though not present in many versions of Forth, it provided a convenient tool when working with double numbers. The comparison destroys both double precision comparands.

D>

d1 d2 --- f

MVP-FORTH UTILITY

True if d1 is greater than d2. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Compare the top two double numbers. True if d1 > d2. (*F-PC*) Same as F83.

Pronounced: d-greater-than

Implementations:

MVP: 8088/86:

: D> DSWAP D< ;

Fig: 8088/86:

Not used.

F83: 8088/86:

: D> 2SWAP D< ;

F-PC:

: D> 2 PICK OVER =

DABS

IF DU< ELSE NIP ROT DROP < THEN ;

Example:

22. 2.3 D>

Enter two double precision values on the stack and compare them. In this case, the resulting flag is true. Note that no decimal point alignment was performed.

Comment: This simple ideogram hardly needs adding to the vocabulary. However, I have often tried it and found it missing, and therefore, have included it among the UTILITY's for symmetry.

D@ addr --- d MVP-FORTH UTILITY

Leave on the stack the contents of the four consecutive bytes beginning at addr, as for a double number. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Pronounced: d-fetch

Implementations:

MVP: 8088/86:

: D@ DUP 2+ @ SWAP @ ;

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

HEX 4AC2 D@ DECIMAL

Take the four bytes beginning at address 4AC2 and place them on the stack. *79S* leaves the storage order within byte pairs unspecified.

Comment: This operation is provided in MVP-FORTH with the ideogram, D@ , to avoid misleading connotations of the numeral 2 . The synonym 2@ may be loaded as an alias for *79S Double Number Word Set* compatibility.

DABS d1 --- d2 MVP-FORTH

Leave as a positive double number d2, the absolute value of a double

number, d1 . Range 0..2,147,483,647 . (*Fig*) Leave the absolute value ud of a double number. (*79S Extension Word Sets*) Same as MVP. (*83S Double Number Extension Word Set*) ud is the absolute value of d. If d is -2,147,483,648 then ud is the same value. See: "arithmetic, two's complement". (*F83*) Return the absolute value of the 32-bit integer on the stack (*F-PC*) Same as F83.

Pronounced: d-abs

Implementations:

MVP: 8088/86:

```
: DABS    DUP    D+-    ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE DABS
  DX POP    DX PUSH    DX DX OR
  ' DNEGATE @-T JS     NEXT
```

F-PC:

```
CODE DABS
  POP AX    OR AX, AX    0>=
  IF 1PUSH  THEN
  POP DX    NEG AX    NEG DX    SBB AX, # 0
  2PUSH    END-CODE
```

Example:

```
2.2 DABS
```

Enter a double precision value and then make its value absolute. In this case it is left unchanged.

Comment: An almost indispensable ideogram when using double precision arithmetic.

DCONSTANT

d ---

MVP-FORTH UTILITY

A defining word used to create a dictionary entry for <name>, leaving d in its parameter field. When <name> is later executed, d will be left on the stack. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Pronounced: d-constant

Form: 2 DCONSTANT <name>

DDROP

Implementations:

MVP: 8088/86:

```
: DCONSTANT CREATE , ,  
DOES> DUP 2+ @ SWAP @ ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
33.33 DCONSTANT NEW-VALUE
```

Enter the value 33.33, which will be a double precision number, and store it in an ideogram named NEW-VALUE. Executing NEW-VALUE will then cause the 3333 to be placed on the stack, the location of the decimal point being lost.

Comment: In MVP-FORTH, the prefix mnemonic D for double precision is used in place of 2 which has the connotation of the numeral value. The synonym 2CONSTANT may be used as an alias in the system for 79S double precision word set compatibility.

DDROP

d ---

MVP-FORTH

Drop the top double number on the stack. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Pronounced: d-drop

Implementations:

MVP: 8088/86:

```
CODE DDROP  
BX POP BX POP NEXT JMP END-COPE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

DDROP

This ideogram will cause the top four bytes on the stack to be removed, which would drop a double precision number or any other four bytes.

Comment: In MVP-FORTH, the prefix mnemonic **D** for double precision is used in place of **2**, which has the connotation of the numeral value. The synonym **2DROP** may be loaded as an alias for *79S Double Number Word Set* compatibility.

DDUP

d --- d d

MVP-FORTH

Duplicate the top double number on the stack. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Pronounced: d-dup

Implementations:

MVP: 8088/86:

CODE DDUP

AX POP DX POP DX PUSH AX PUSH
DPUSH JMP END-CODE

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

33.33 DDUP

Place the double precision number on the stack and duplicate it.

Comment: In MVP-FORTH, the prefix mnemonic **D** for double precision, is used in place of **2** which has the connotation of the numeral value. The synonym **2DUP** may be loaded as an alias for *79S Double Number Word Set* compatibility.

DECIMAL

MVP-FORTH

Set the input-output numeric conversion base to ten. (*Fig*) Set the

DEFER

numeric conversion BASE for decimal input-output. (79S) Same as MVP. (83S) Set the input-output numeric conversion base to ten. (F83) All subsequent numeric I-O will be in Decimal. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

HEX

: DECIMAL 0A BASE ! ;

Fig: 8088/86:

: DECIMAL 10 BASE ! ;

F83: 8088/86:

Same as Fig.

F-PC:

Same as Fig.

Example:

DECIMAL

Using this ideogram assures you that the present value of BASE is decimal 10 .

Comment: This ideogram DECIMAL lets you put the value ten into BASE without having to know its current value. Note that " 10 BASE ! " is useless regardless of the base you are currently using. Except for the disk interface, all of the implementation code uses HEX for the number base.

DEFER | <name> --- NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Defining word for execution vectors. These are initially set to display an error message. They are initialized with IS. (F-PC) Define a vectored execution word. These are initially set to display an error message. They are initialized with IS.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

DEFINITIONS

```
: DEFER      CREATE    [ ' ] CRASH ,    ;USES    DODEFER ,  
F-PC:  
: DEFER      CREATE    [ ' ] CRASH ,    ;USES    DODEFER , -  
X
```

Example:

```
DEFER TEST
```

Execution of TEST will do nothing. Some implementations will return an error message that the execution vector has not been initialized.

Comment: The ideogram is not in any standard, but has been implemented in a number of Forth dialects. It provides for forward referencing or changing of functions to be defined later.

DEFINITIONS

MVP-FORTH

Set CURRENT to the CONTEXT vocabulary so that subsequent definitions will be created in the vocabulary previously selected as CONTEXT. (*Fig*) Used in the form: cccc DEFINITIONS. Set the CURRENT vocabulary to the CONTEXT vocabulary. In the example, executing the vocabulary name cccc made it the CONTEXT vocabulary and executing DEFINITIONS made both specify vocabulary cccc. (79S) Same as MVP. (83S) The compilation vocabulary is changed to be the same as the first vocabulary in the search order. See: "vocabulary, compilation". (F83) Subsequent definitions will be placed into CURRENT (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

```
: DEFINITIONS    CONTEXT @    CURRENT !    ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Same as MVP.

F-PC:

Same as MVP.

Example:

```
FORTH DEFINITIONS
```

This sets the necessary pointer so that new definitions will be added to the Forth vocabulary.

DENSITY

Comment: This ideogram is necessary in starting a new vocabulary or adding to an existing one. When using Ragsdale's vocabulary modifications the implementation will have to be modified.

DEN

MVP-FORTH DISK I-O

An array of variables which stores a value of the variable, DENSITY, for each of the drives up to a maximum of 5. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

VARIABLE DEN 0 DEN 1 0 , 1 , 1 , 1 ,

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

1 2* DEN + @ .

To select a value for DENSITY on drive 1, enter a value 1 and then calculate the offset from the address of DEN to the location of the current density code for that drive. Then, fetch and print that value.

Comment: The values are initialized with a COLD start and can be subsequently changed according to the responses to CONFIGURE. The values associated with DENSITY are encoded as integers according to the prompts given when CONFIGURE is executed.

DENSITY

--- addr

MVP-FORTH DISK I-O

A variable used by the disk interface. It may have a value from 0 to 6 according to the particular drive being selected. (*Fig*) Not defined. (*Fig - 8086/88 only*) A variable that contains the current disk density. Zero = single density (26 sectors/track). Non-zero = double density (52 sectors/track). Sector size is 128 bytes in both densities. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

VARIABLE DENSITY 0 DENSITY 1

Fig: 8088/86:

0 VARIABLE DENSITY

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

DENSITY @ .

The address of the variable DENSITY is placed on the stack. The value of the variable is then fetched and printed.

Comment: In the MVP-FORTH implementation, the original values associated with the variable in fig-Forth are extended to include a number of possible configurations. For each value a different number of sectors per track and tracks per disk are taken from the appropriate arrays and used in T&SCALC in making the necessary calculations to access the proper physical sector on the disks. The values are modified from those used by fig-Forth to run with the current interface.

DEPTH

--- n

MVP-FORTH

Leave the number of the quantity of 16-bit values contained in the data stack, before n was added. (*Fig*) Not defined. (*79S*) Same as MVP. (*83S*) +n is the number of 16-bit values contained in the data stack before +n was placed on the stack. (*F83*) Returns the number of items on the parameter stack. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

: DEPTH SP@ S0 SWAP - 2/ ;

Fig: 8088/86:

Not used.

F83: 8088/86:

: DEPTH SP@ SP0 @ SWAP - 2/ ;

F-PC:

Same as F83.

DIGIT

Example:

DEPTH . 7

Find the number of values on the stack and print it.

Comment: This ideogram is the 79S higher level alternative to calculations which can be made with S0 and SP@. It is more general and less implementation dependent, but it is not defined in fig-Forth and some other systems.

DIGIT

MVP-FORTH

```
c  n1  ---  n2  tf
c  n1  ---  ff          (bad)
```

Converts the ASCII character c (using base n1) to its binary equivalent n2, accompanied by a true flag. If the conversion is invalid, leaves only a false flag. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Returns a flag indicating whether or not the character is a valid digit in the given base. If so, returns converted value and true, otherwise returns char and false. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

CODE DIGIT

```
DX POP    AX POP    0 AL, SUB    DIGI2 JB
9 AL, CMP  DIGI1 JBE    7 AL, SUB
0A AL, CMP  DUGI2 JB
HERE LABEL DIGI1    DL AL, CMP  DIGI2 JAE
DX DX, SUB  AL DL, MOV  1 AL, MOV  DPUSH JMP
AX AX, SUB
HERE LABEL DIGI2    AX AX, SUB
APUSH JMP  END-CODE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

CODE DIGIT

```
DX POP    AX POP    AX PUSH  ASCII 0 # AL SUB
FAIL JB   9 # AL CMP  >
IF  17 # AL CMP  FAIL JB   7 # AL SUB  THEN
DL AL CMP  FAIL JAE  AL DL MOV  AX POP
TRUE # AX MOV  2PUSH
```

F-PC:

CODE DIGIT

```

    POP DX    POP AX    PUSH AX    SUB AL, # ASCII 0
    JB 0 $    CMP AL, # 9 >
    IF    CMP AL, # 17    JB 0 $    SUB AL, # 7    THEN
    CMP AL, DL    JAE 0 $    MOV DL, AL    POP AX
    MOV AX, # TRUE    2PUSH
    0 $:    SUB AX, AX    1PUSH    END-CODE

```

Example:

```

: CONVERT
  BEGIN    1+    DUP    >R    C@    BASE    @    DIGIT
  WHILE    SWAP    BASE    @    U*    DROP    ROT    BASE    @
    U*    D+    DPL    @    1+
    IF    1 DPL +! THEN    R
  REPEAT    R> ;

```

In this example taken from the source code of MVP-FORTH, CONVERT calls DIGIT to test whether the ASCII character on the stack is a valid numeric.

Comment: It is a primitive in most versions of Forth.

DISK-ERROR

--- addr

MVP-FORTH DISK I-O

A variable used by the disk interface, containing the disk status for the last sector read or written. 0 means no error. (*Fig*) Not defined. (*Fig - 8086/88 only*) A variable that contains disk error status (non-zero indicates a disk error). The disk status is saved after each sector read/write but error trapping has not been implemented (the error status could be tested in "R/W"). (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Returns the address of a variable which contains error information on the most recent attempt to access the disk.

Implementations:

MVP: 8088/86:

```
VARIABLE DISK-ERROR    0 DISK-ERROR !
```

Fig: 8088/86:

```
0 VARIABLE DISK-ERROR
```

F83: 8088/86:

```
VARIABLE DISK-ERROR
```

F-PC:

Same as F83.

DLITERAL

Example:

```
DISK-ERROR @ .
```

Check the value of the variable by fetching it and printing it.

Comment: This word makes it possible to check for disk errors from Forth. After calling the I-O routines via SEC-READ or SEC-WRITE, this variable contains the byte error code returned by the operating system. See the system manual for details if you wish to implement action based on the error codes.

DLIST

NOT USED IN MVP-FORTH

(Fig) List the names of the dictionary entries in the CONTEXT vocabulary. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

(This ideogram has not been implemented.)

Comment: This ideogram differs from VLIST by listing only the CONTEXT vocabulary without chaining to Forth. It is not used in most fig-Forth implementations.

DLITERAL

MVP-FORTH

```
      d --- d    (executing )
      d ---      (compiling)
```

If compiling, compile a stack double number into a literal. Later execution of the definition containing this literal will push it to the stack. If executing, the number will remain on the stack. (Fig) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Both LITERAL and DLITERAL are TRANSITION words, ie IMMEDIATE Compile the double integer from the stack as a literal. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

```
: DLITERAL  STATE @
  IF SWAP [COMPILE] LITERAL
    [COMPILE] LITERAL THEN ; IMMEDIATE
```

Fig: 8088/86:

```
: DLITERAL  STATE @
  IF SWAP [COMPILE] LITERAL
    [COMPILE] LITERAL ENDIF ; IMMEDIATE
```

F83: 8088/86:

```
: DLITERAL
  SWAP [COMPILE] LITERAL [COMPILE] LITERAL ;
IMMEDIATE
```

F-PC:

Same as F83.

Example:

```
: INTERPRET
  BEGIN -FIND
    IF STATE @
      IF 2- , ELSE 2- EXECUTE THEN ?STACK
    ELSE HERE NUMBER DPL @ 1+
      IF [COMPILE] DLITERAL
        ELSE DROP [COMPILE] LITERAL
        THEN ?STACK
      THEN
    AGAIN ;
```

This example comes from an earlier version of the MVP-FORTH source code.

Comment: Though not always present in Forth it is usually necessary for any serious double precision arithmetic.

DMAX d1 d2 --- d3 MVP-FORTH UTILITY

Leave the larger of two double numbers. (*Fig*) Not defined. (*79S Extension Word Sets*) Same as MVP. (*83S Double Number Extension Word Set*) d3 is the greater of d1 and d2. (*F83*) Return the greater of the top two double numbers. (*F-PC*) Same as F83.

Pronounced: d-max

Implementations:

MVP: 8088/86:

```
: DMAX DOVER DOVER D< IF DSWAP THEN
  DDROP ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: DMAX 4DUP D< IF 2SWAP THEN 2DROP ;
```

F-PC:

DMINUS

: DMAX 2DUP D< IF 2SWAP THEN 2DROP ;
Note: This code is in error; Should be same as F83.

Example:

2.2 3.3 DMAX

Enter the two double precision values and then test them, leaving the larger on the stack. In this case the value 33 as a double precision, 4 byte value is left. Note that the decimal point is not considered.

Comment: A convenient double precision operator.

DMIN d1 d2 --- d3 MVP-FORTH UTILITY

Leave the smaller of two double numbers. (*Fig*) Not defined. (79S *Extension Word Sets*) Same as MVP. (83S *Double Number Extension Word Set*) d3 is the lesser of d1 and d2. (F83) Return the lesser of the top two double numbers. (F-PC) Same as F83.

Pronounced: d-min

Implementations:

MVP: 8088/86:

```
: DMIN DOVER DOVER D< NOT
  IF DSWAP THEN DDROP ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: DMIN 4DUP D> IF 2SWAP THEN 2DROP ;
```

F-PC:

Same as F83.

Example:

2.2 3.3 DMIN

Enter two double precision values and then test them, leaving the smaller on the stack. In this case the double precision value 22 is left in the top 4 bytes on the stack. Note that the decimal point is not considered.

Comment: A convenient double precision operator.

DMINUS d1 --- d2 NOT USED IN MVP-FORTH

(*Fig*) Convert d1 to its double number two's complement. (79S) Not

defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86: Could be defined as:
 : DMINUS DNEGATE ;

Fig: 8088/86:
 CODE DMINUS AX POP AX NEG APUSH JMP

F83: 8088/86:
 Not used.

F-PC:
 Not used.

Example:

2.2 DMINUS

Enter the double precision value on the stack and then change the sign.
 In this case, the double precision value -22 is left on the stack.

Comment: This ideogram is now obsolete and its function is replaced
 with DNEGATE . Furthermore, its pronunciation would clash with D- .
 It may appear in older programs.

DNEGATE

d1 --- -d1

MVP-FORTH

Leave the two's complement of a double number. (Fig) Not defined.
 (79S) Same as MVP. (83S) d2 is the two's complement of d1. (F83)
 Same as NEGATE except for double precision numbers. (F-PC) Same
 as F83.

Pronounced: d-negate

Implementations:

MVP: 8088/86:
 CODE DNEGATE
 BX POP CX POP AX AX, SUB AX DX, MOV
 CX DX, SUB BX AX, SBB DPUSH JMP END-CODE

Fig: 8088/86:
 Not used.

F83: 8088/86:
 CODE DNEGATE
 BX POP CX POP AX AX SUB AX DX MOV
 CX DX SUB BX AX SBB 2PUSH

DO

F-PC:

```
CODE DNEGATE
  POP AX    POP DX    NEG AX    NEG DX
  SBB AX, # 0    2PUSH    END-CODE
```

Example:

```
2.2 DNEGATE
```

Enter the double precision value on the stack and then leave the negative of its value. In this case, the double precision value -22 is left on the stack.

Comment: This ideogram replaces the now obsolete one, DMINUS .

DO n1 n2 --- MVP-FORTH

Use in a colon definition. Begin a loop which will terminate based on control parameters. The loop index begins at n2, and terminates based on the limit n1. At LOOP or +LOOP, the index is modified by a positive or negative value. The range of a DO-LOOP is determined by the terminating word. DO-LOOP may be nested. Capacity for three levels of nesting is specified as a minimum for standard systems. (*Fig*) Occurs in a colon-definition ... At run time, DO begins a sequence with a repetitive execution controlled by a loop limit n1 and an index with initial value n2. DO removes these from the stack. Upon reaching LOOP the index is incremented by one. Until the new index equals or exceeds the limit, execution loops back to just after DO; otherwise the loop parameters are discarded and execution continues ahead. Both n1 and n2 are determined at run-time and may be the result of other operations. Within a loop, I will copy the current value of the index to the stack. See I, LOOP, +LOOP, LEAVE. When compiling within the colon-definition, DO compiles (DO), leaves the following address addr and n for later error checking. (*79S*) Same as MVP. (*83S*) Begins a loop which terminates based on control parameters. The loop index begins at n2, and terminates based on the limit n1. See LOOP and +LOOP for details on how the loop is terminated. The loop is always executed at least once. For example: n DUP DO ... LOOP executes 65,536 times. sys is balanced with its corresponding LOOP or +LOOP. See: "9.9 Control Structures". An error condition exists if insufficient space is available for at least three nesting levels. (*F83*) Not defined. (*F-PC*) Initialize a loop structure with index running from start to limit-1.

Form:

```
DO ... LOOP
DO ... n +LOOP
```


MVP: 8088/86:

```
: DO    COMPILE  <DO>  HERE  3  ;  IMMEDIATE
```

Fig: 8088/86:

```
: DO    COMPILE    (DO)    HERE  3  ;  IMMEDIATE
```

F83: 8088/86:

```
: DO    COMPILE (DO)    >MARK 3000  ;  IMMEDIATE
```

F-PC:

```
: DO    COMPILE (DO)    ?>MARK ;  IMMEDIATE
```

Example:

```
: TEST    10  1  DO  I  .  LOOP  ;
```

This definition used the DO structure to print the digits 1 through 9.

Comment: DO-LOOPS are one of the three major control structures (IF , BEGIN , DO) in Forth. They are more efficient than BEGIN loops when an index must be incremented and compared with a limit. The ideogram LEAVE may be used to terminate a DO-LOOP before the index has run its full course. This implementation keeps the loop limits and indices on the return stack. For indexing on addresses, use a step value with /LOOP.

DOES>

MVP-FORTH

Define the run-time action of a word created by a high-level defining word. It marks the termination of the defining part of the defining word <name> and begins the definition of the run time action for words that will later be defined by <name>. On execution of <name> the sequence of words between DOES> and ; will be executed, with the address of <name>'s parameter field on the stack. (Fig) A word which defines the run-time action within a high-level defining word. DOES> alters the code field and first parameter of the new word to execute the sequence of compiled word addresses following DOES>. Used in combination with <BUILDS. When the DOES> part executes it begins with the address of the first parameter of the new word on the stack. This allows interpretation using this area or its contents. Typical uses include the Forth assembler, multidimensional arrays, and compiler generation. (79S) Same as MVP. (83S) Defines the execution-time action of a word created by a high-level defining word. Used in the form ... [Following] any user defining word which executes CREATE. Marks the termination of the defining part of the defining word <name> and then begins the definition of the execution-time action for words that will later be defined by <name>. When <name> is later executed, the address of

DOES>

<name>'s parameter field is placed on the stack and then the sequence of words between DOES> and ; are executed. (F83) Compile the code field for (;CODE) and a CALL instruction to the run time for DOES>, called DODOES. Specifies the run time of a defining word in high level Forth. (F-PC) Same as F83.

Form; : <name> ... CREATE ... DOES> ... ;
and then, <name> <namex>

Pronounced: does

Implementations:

MVP: 8088/86:

HEX

```
: DOES>
  ?CSP COMPILE :CODE E8 C, DODOES HERE 2+ - , ;
  HERE LABEL DODOES ASSEMBLER
  DX INC BP DEC BP DEC SI [BP] MOV
  SI POP DX PUSH NEXT JMP FORTH
```

Fig: 8088/86:

```
: DOES> R> LATEST PFA ! ( ;CODE)
  HERE LABEL DODOE SP BP, XCHG
  SI PUSH SP BP, XCHG DX INC DX BX, MOV
  [BX] SI, MOV DX INC DX INC
  DX PUSH NEXT JMP
```

F83: 8088/86:

```
: DOES>
  COMPILE ( ;CODE) 232 ( CALL ) C,
  [ DODOES ] LITERAL HERE 2+ - , ;
  IMMEDIATE
```

F-PC:

```
: DOES>
  COMPILE ( ;CODE) HERE X, 232 ( CALL ) C,
  [ [FORTH] ASSEMBLER DODOES META ] LITERAL
  HERE 2+ - , XHERE PARAGRAPH + DUP XDPSEG !
  XSEG @ - , XDP OFF ; IMMEDIATE
```

Example:

```
: DCONSTANT CREATE SWAP , ,
DOES DUP @ SWAP 2+ @ ;
```

This example comes from the implementation of MVP-FORTH utilities.

Comment: This implementation uses the 79S CREATE ... DOES> technique which differs internally from fig-Forth's now obsolete <BUILDS ... DOES> . DOES> is immediate. It compiles ;CODE and a machine language call to the low level routine labeled "DODOES". By embedding that one machine instruction in the body of the defining word, two bytes are saved from every one of its generated offspring. But perhaps more importantly, the 79S approach makes ' (tick) work on DOES> products without the inconsistent adjustment by two required in fig-Forth.

DONE?

n --- f

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) True if the input stream is exhausted or state doesn't match.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

CODE DONE?

POP AX CMP AX, STATE 0<>

IF MOV END? # 0 WORD MOV AX, # -1 1PUSH

THEN PUSH END? MOV END? # 0 WORD

NEXT END-CODE

Example:

: INTERP

BEGIN ?STACK DEFINED

IF EXECUTE ELSE NUMBER DOUBLE? NOT

IF DROP THEN

THEN FALSE DONE?

UNTIL ;

The default Forth interpret loop in F-PC.

Comment: Most implementations of Forth need some way to determine the end of the input stream. MVP-FORTH uses ?STREAM.

DP

DOVER d1 d2 --- d2 d1 MVP-FORTH UTILITY

Leave a copy of the second double number on the stack. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Pronounced: d-over

Implementations:

MVP: 8088/86:

 : DOVER 4 PICK 4 PICK ;

Fig: 8088/86:

 Not used.

F83: 8088/86:

 Not used.

F-PC:

 Not used.

Example:

 33.33 44.44 DOVER

Place the two double precision numbers on the stack and then place a copy of the first one on top.

Comment: The prefix mnemonic D for double precision is used in place of 2 which has the connotation of the numeral value. However, its synonym, 2OVER, may be loaded as an alias for compatibility with the *79S Double Precision Number Word Set*.

DP --- addr MVP-FORTH

A user variable, the dictionary pointer, which contains the address of the next free memory above the dictionary. The value may be read by HERE and altered by ALLOT. (*Fig*) Same as MVP. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Size of dictionary. Next available location. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

 HEX 12 USER DP

Fig: 8088/86:

 Same as MVP.

F83: 8088/86:

VARIABLE DP

F-PC:

Same as F83.

Example:

DP @ U.

Entering the user variable name leaves the address of the variable which can then be fetched and printed as an unsigned number.

Comment: This is a primitive in most implementations of Forth even if it is not immediately available to the programmer. Some implementations may give it the name H.

DPL

--- addr

MVP-FORTH

A user variable containing the number of digits to the right of the decimal on double integer input. It may also be used to hold output column location of a decimal point, in user generated formatting. The default value on single number input is -1. (*Fig*) Same as MVP. (*79S Reference Word Set*) A variable containing the number of places after the fractional point for output conversion. If DPL contains zero, the last character output will be a decimal point. No point is output if DPL contains a negative value. DPL may be set explicitly, or by certain output words, but is unaffected by number input. (*83S*) Not defined. (*F83*) The decimal point location for number input. (*F-PC*) Same as F83.

Pronounced: d-p-l

Implementations:

MVP: 8088/86:

HEX 42 USER DPL

Fig: 8088/86:

HEX 28 USER DPL

F83: 8088/86:

VARIABLE DPL

F-PC:

Same as F83.

Example:

2.2 DPL @ .

Entering a double precision value, followed by fetching the value at

DPUSH

this user variable and printing it, will show the number of digits which were entered to the right of the decimal point; in this case, 1 .

Comment: This ideogram may save you expense and worry with a floating point package. User-defined numeric input routines may inspect the value of DPL and adjust the converted number as necessary. This makes the use of scaled, fixed point arithmetic transparent to the user who, for example, need not type in unnecessary trailing zeros after a decimal point.

DPUSH

--- addr

MVP-FORTH

A constant used in 8080/Z80 and 8086/8088 implementations pointing to a machine code entry point which pushes the DE or DX register followed by the HL or AX register onto the stack and then falls through to NEXT, the inner interpreter. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

DPUSH CONSTANT DPUSH

Fig: 8088/86:

Not used.

F83: 8088/86:

Not defined.

F-PC:

Not defined.

(The cross-compiler takes the label DPUSH and makes it a constant.
)

Example:

```
CODE DDUP
  H POP  D POP  D PUSH  H PUSH
  DPUSH JMP  END-CODE
```

The exmple of 8080 code is from the MVP-FORTH source.

Comment: This is a machine, system and implementation dependent location used only at the machine code level. It allows trimming two bytes off CODE definition which would otherwise end with: D PUSH H PUSH NEXT JMP.

DR-DEN

n1 --- n2

MVP-FORTH DISK I-O

Converts the drive number to the current density code for that drive. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

: DR-DEN 2* DEN + @ ;

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example

2 DR-DEN

Enter the number for a given drive and the ideogram will convert the value on the stack to the current code value of **DENSITY**.

Comment: The calculation is made by reference to the array **DEN**. Nothing is actually fetched from or stored into the variable **DENSITY** with this ideogram.

DRO

MVP-FORTH DISK I-O

DR1**DR2****DR3****DR4**

Installation dependent commands to select disk drives, by presetting **OFFSET**. The contents of **OFFSET** is added to the block number in **BLOCK** to allow for this selection. (*Fig*) Same as MVP with the following added: Offset is suppressed for error test so that it may always originate from drive 0. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

DRIVE

```
: DR0    0  OFFSET  !  ;
: DR1    DR0  0  SET-DRX  ;
: DR2    DR1  1  SET-DRX  ;
: DR3    DR2  2  SET-DRX  ;
: DR4    DR3  3  SET-DRX  ;
```

Fig: 8088/86:

HEX

```
: DR0    0  OFFSET  !  ;
: DR1    DENSITY @
      IF 0FA4 ELSE 07D2 ENDIF OFFSET ! ;
NOTE: Values are system dependent!
```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

DR0

Entering this ideogram sets the value of the variable OFFSET to 0 .

Comment: These ideograms affect only the value in OFFSET. Because they use the current disk configuration parameters in their calculation, they are safer, easier, and more flexible than storing a value directly into OFFSET. More than five drives cannot be accommodated without re-cross-compiling MVP-FORTH.

DRIVE

--- addr

MVP-FORTH DISK I-O

A variable used by the disk interface, containing the disk drive number (0 to MAX-DRV) used on the last sector read or written. (*Fig*) Not defined. (*Fig - 8086/88 only*) A variable that contains the current disk drive number. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
VARIABLE DRIVE    0 DRIVE !
```

Fig: 8088/86:

```
0 VARIABLE DRIVE
```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
1 DRIVE !
```

Enter the desired drive number, in this case 1 , and store it in the variable designated by this ideogram.

Comment: This ideogram SET-DRIVE passes the value of this variable to the operating system to select a physical drive. Ordinarily, this value should be no greater than the value of the alterable constant, MAX-DRV, less one.

DROP

n ---

MVP-FORTH

Drop the top number from the stack. (*Fig*) Same as MVP. (*79S*) Same as MVP. (*83S*) 16b is removed from the stack. (*F83*) Throw away the top element of the stack. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

```
CODE DROP
  AX POP    NEXT JMP    END-CODE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE DROP
  AX POP    NEXT
```

F-PC:

```
CODE DROP
  ADD SP, # 2    NEXT    END-CODE
```

Example:

```
45 DROP
```

Enter a number and then drop it as though nothing happened.

Comment: A common ideogram used in virtually all implementations of Forth.

DSWAP

d1 d2 --- d2 d1

MVP-FORTH UTILITY

Exchange the top two double numbers on the stack. (*Fig*) Not defined.

DU<

(79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Pronounced: d-swap

Implementations:

MVP: 8088/86:

```
: DSWAP 4 ROLL 4 ROLL ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
33.33 44.44 DSWAP
```

Place two double precision numbers on the stack and then exchange their order. The location of the decimal point is ignored.

Comment: The prefix mnemonic D for double precision is used in place of 2, which has the connotation of the numeral value. However, its synonym, 2SWAP, may be loaded as an alias for compatibility with the 79S Double Precision Number Word Set.

DU<

ud1 ud2 --- flag

MVP-FORTH UTILITY

True if ud1 is less than ud2. Both numbers are unsigned. (Fig) Not defined. (79S Extension Word Sets) Same as MVP. (83S Double Number Extension Word Set) flag is true if ud1 is less than ud2. Both numbers are unsigned. (F83) Performs unsigned comparison of two double numbers. (F-PC) Same as F83.

Pronounced: d-u-less

Implementations:

MVP: 8088/86:

```
: DU< >R >R 8000 +  
R> R> 8000 + D< ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```

: DU<   ROT   SWAP   2DUP   U<
  IF     2DROP 2DROP TRUE
  ELSE  <>
    IF   2DROP FALSE ELSE U< THEN
  THEN ;

```

F-PC:

```

CODE DU<
  POP DX   POP BX   POP CX   POP AX
  SUB AX, BX   SBB CX, DX   SBB AX, AX
  1PUSH   END-CODE

```

Example:

```
3.3  2.2  DU<
```

Enter two double precision values and compare them as unsigned numbers, in this case, leaving a false flag of 0 on the stack. Note that the location of the decimal point is not considered.

Comment: With some double precision operations unsigned numbers are important and this comparison is necessary. Note that both double precision comparands are destroyed by the comparison.

DUMP

addr n ---

MVP-FORTH UTILITY

List the contents of n addresses starting at addr. Each line of values may be preceded by the address of the first value. (*Fig*) Print the contents of n memory locations beginning at addr. Both addresses and contents are shown in the current numeric base. (*79S Reference Word Set*) Same as MVP. (*83S*) Not defined. (*F83*) A primitive little dump routine to help you debug after you have changed the system source and nothing works any more. (*F-PC*) Dump an area of the Code segment.

Implementations:

MVP: 8088/86:

```

HEX
: DUMP
  BASE @ >R HEX   DUMP-HEADER 0
  DO CR DUP I + DUP 0 9
    D.R 2 SPACES DUP 8 0
    DO DUP I + C@ 3 .R LOOP
    DROP SPACE DUP 8 + 8 0
    DO DUP I + C@ 3 .R LOOP
    DROP 3 SPACES 10 0

```

DUP

```
DO DUP I + C@
  DUP 20 < OVER 7E > OR
  IF DROP 2E THEN EMIT
LOOP DROP 10
PAUSE
?TERMINAL IF LEAVE THEN
/LOOP
DROP CR R> BASE ! ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: DUMP
  BASE @ -ROT HEX .HEAD BOUNDS
  DO I DLN KEY? ?LEAVE 16
+LOOP BASE !
```

F-PC:

```
: DUMP ?CS: -ROT LDUMP ;
```

Example:

```
HEX
100 80 DUMP
DECIMAL
```

Dump the first 128 bytes starting from the normal origin of Forth.

Comment: A variety of implementations of this ideogram generate various formats. Some of them include printing the ASCII characters where printable or a dot if they are not. The MVP-FORTH version incorporates a PAUSE feature, by which pressing any key will freeze the display. Once suspended, the DUMP may be resumed by a single keystroke, or aborted by striking any two keys in rapid succession.

DUP

n --- n n

MVP-FORTH

Leave a copy of the top stack number. (*Fig*) Duplicate the value on the stack. (*79S*) Same as MVP. (*83S*) Duplicate 16b. (*F83*) Duplicate the top element of the stack. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

```
CODE DUP AX POP AX PUSH APUSH JMP END-CODE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

CODE DUP AX POP AX PUSH 1PUSH

F-PC:

CODE DUP

MOV DI, SP PUSH 0 [DI] NEXT END-CODE

Example:

45 DUP

Enter the value 45 on the stack and then duplicate its value on the top of the stack.

Comment: A common ideogram present in virtually all implementations of Forth.

DVARIABLE

MVP-FORTH UTILITY

A defining word used to create a dictionary entry of <name> and assign 4 bytes for storage in the parameter field. When <name> is later executed, it will leave the address of the first byte of its parameter field on the stack. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Form: DVARIABLE <Name>

Implementations:

MVP: 8088/86:

DVARIABLE CREATE 4 ALLOT ;

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

DVARIABLE NEW-VALUE

Make a new ideogram referring to a double precision variable. Its value is not initialized.

Comment: The prefix mnemonic D for double precision is used in place of 2, which has the connotation of the numeral value. However, its

ELSE

synonym, 2VARIABLE, may be loaded as an alias for compatibility with the 79S *Double Precision Number Word Set*.

EDITOR

MVP-FORTH

The name of the editor vocabulary. When this name is executed, EDITOR is established as the CONTEXT vocabulary. (*Fig*) Not defined. (*79S Reference Word Set*) Same as MVP. (*83S*) Not defined. (*F83*) Not defined. (*F83*) Not defined. (F-PC) - The vocabulary that contains all of the editor words. Normally you use "line# EDIT", "SED filename" or just "ED" to start the editor.

Implementations:

(An editor is not included as a part of the MVP-FORTH source code; however, the line editor from Forth Dimensions is added as a vocabulary to the fully configured Forth binary image.)

Comment: A variety of EDITORS is available. They are of various degrees of complexity. The *Installation Manual* of fig-Forth includes a simple version. A version which performs all of the functions as defined in *Starting Forth* is available in *Forth Dimensions*. The latter can be loaded directly with fig-Forth and, after minor modifications, with MVP-FORTH. A Screen Editor has been published in *Dr. Dobbs Journal* which can be loaded with either fig-Forth or 79S with appropriate changes. The full MVP-FORTH binary file includes the EDITOR by Sam Daniels adapted from *Forth Dimensions*, Vol III No. 3. It complies with the EDITOR description in *Starting Forth*.

Note: The EDITOR is a separate vocabulary. It must be called before trying the EDITOR's ideograms as described in Chapter 3 of *Starting Forth*.

ELSE

MVP-FORTH

Used in a colon-definition and executes after the true part following IF ... ELSE forces execution to skip till just after THEN. It has no effect on the stack. (See IF) (*Fig*) Occurs within a colon-definition in the form ... At run-time, ELSE executes after the true part following IF. ELSE forces execution to skip over the following false part and resumes execution after the ENDIF. It has no stack effect. (*79S*) Same as MVP. (*83S*) Used in the form: ... ELSE executes after the true part following IF. ELSE forces execution to continue at just after THEN. sys1 is balanced with its corresponding IF. sys2 is balanced with its corresponding THEN. See: IF ... THEN. (*F83*) Not defined. (F-PC) Used in the form: flag IF <true> ELSE <false> THEN. If flag is false,

branches forward to <false>.

Form: IF ... ELSE ... THEN

Implementations:

MVP: 8088/86:

```
: ELSE 2 ?PAIRS COMPILE BRANCH HERE 0 ,
  SWAP 2 [COMPILE] THEN 2 ; IMMEDIATE
```

Fig: 8088/86:

```
: ELSE 2 ?PAIRS COMPILE BRANCH HERE 0 , SWAP
2
  [COMPILE] ENDIF 2 ; IMMEDIATE
```

F83: 8088/86:

```
: ELSE COMPILE BRANCH >MARK 1000
  2SWAP [COMPILE] THEN ; IMMEDIATE
```

F-PC:

```
: ELSE COMPILE BRANCH ?>MARK 2SWAP ?>RESOLVE ;
IMMEDIATE
```

Example:

```
: TEST ?DUP IF . ELSE ." zero " THEN ;
```

This definition will test the value on the top of the stack and if it is 0 , will print the word 'zero' in place of the number.

Comment: In keeping with the traditions of structured programming, the ELSE clause is optional, and the IF ... ELSE ... THEN constructs may be nested. Note that IF ... ELSE ... ELSE ... THEN will go untrapped in this implementation.

EMIT

c ---

MVP-FORTH

Transmit character to the current output device. (*Fig*) Transmit ASCII character *c* to the selected output device. *OUT* is incremented for each character output. (*79S*) Same as MVP. (*83S*) The least-significant 7-bit ASCII character is displayed. See "9.5.3 EMIT". (*F83*) Sends a character to the output device. (*F-PC*) A deferred word which sends a character to the output device.

Implementations:

MVP: 8088/86:

```
: EMIT 'EMIT @ EXECUTE ;
```

Fig: 8088/86:

EMPTY

```
: EMIT  Pemit 1 OUT P! ;
```

F83: 8088/86:

```
: EMIT  EMITCHAR C! EMITCHAR 1 TYPE ;
```

F-PC:

```
DEFER EMIT
```

Example:

```
65 EMIT
```

Entering the decimal value 65 followed by this ideogram will cause the character A to be printed.

Comment: All Forth terminal output is handled through EMIT. Installation dependencies, though unavoidable, are at least confined within this ideogram. MVP-FORTH builds a vector into EMIT so that output may be redirected to printers, alternate terminals, or even operating system files. By default, EMIT vectors to <EMIT>.

EMPTY

MVP-FORTH SUPPLEMENTAL

Forget all new words added to the dictionary by the user. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined

Implementations:

MVP: 8088/86:

```
HEX
```

```
: EMPTY  INIT-FORTH @ ' FORTH 2+ !  
      INIT-USER UP @ 6 + 30 CMOVE ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: MARK  
      CREATE DOES> (FORGET) FORTH DEFINITIONS ;  
MARK EMPTY  HERE FENCE !
```

F-PC:

Not used.

Example:

```
EMPTY
```

This ideogram will reconfigure the system to the initial values at the time of start up, or the last FREEZE.

Comment: This ideogram does essentially the same thing as COLD i.e. restarts the Forth system. The only difference is that the stack is not cleared, and the block buffers are unaffected.

EMPTY-BUFFERS

MVP-FORTH

Mark all block buffers as empty, without necessarily affecting their actual contents. UPDATED blocks are not written to mass storage. (*Fig*) Mark all block-buffers as empty, not necessarily affecting the contents. Updated blocks are not written to disc. This is also an initialization procedure before first use of the disc. (79S) Same as MVP. (83S) Not defined. (F83) First wipe out the data in the buffers. Next initialize the buffer pointers to point to the right addresses in memory and set all of the update flags to unmodified. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: EMPTY-BUFFERS  FIRST  LIMIT  OVER  - 0 <FILL>
  #BUFF 0
  DO 7FFF HDBT I * FIRST + ! LOOP ;
```

Note: HDBT is compiled to a hex literal value 404 which is the number of bytes in a block buffer plus four.

Fig: 8088/86:

```
: EMPTY-BUFFERS  FIRST  LIMIT  OVER  - ERASE ;
```

F83: 8088/86:

```
: EMPTY-BUFFERS
  FIRST LIMIT OVER - ERASE
  >BUFFERS #BUFFERS 1+ 8* ERASE
  FIRST 1 BUFFER# #BUFFERS 0
  DO DUP ON 4 + 2DUP ! SWAP B/BUF + SWAP 4 +
  LOOP 2DROP ;
```

F-PC:

Not used.

Example:

EMPTY-BUFFERS

This ideogram will empty the contents of all buffers in memory so that those marked for UPDATE will also be erased and not written back to disk.

Comment: In this implementation, EMPTY-BUFFERS clears all buffers to nulls.

ENCLOSE

CAUTION — Think twice before using it, to make sure you aren't destroying irreplaceable data.

ENCLOSE `addr1 c --- addr1 n1 n2 n3 MVP-FORTH`

The text scanning primitive used by WORD. From the text address `addr1` and an ASCII delimiting character `c`, is determined [by] the byte offset to the first non-delimiter character `n1`, the offset to the first delimiter after the text `n2`, and the offset to the first character not included `n3`. This procedure will not process past an ASCII 'null', treating it as an unconditional delimiter. (*Fig*) Same as MVP. (*Fig 8088/86 Modified*) Modified to return 16-bit offset values for `n1`, `n2` and `n3`. The Fig model only returns 8-bit offsets which could limit the range of word searches in blocks larger than 256 bytes. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

CODE ENCLOSE

```
AX POP    BX POP    BX PUSH    0FFFF DX, MOV    BX DEC
HERE LABEL ENCL1    BX INC    DX INC    [BX] AL, CMP
ENCL1 JZ    DS PUSH    [BX] PTR BYTE INC
[BX] PTR BYTE DEC    ENCL2 JNZ
DX INC    DX PUSH    DX DEC    DX PUSH    NEXT JMP
HERE LABEL ENCL2    BX INC    DX INC    [BX] AL, CMP
ENCL3 JZ    [BX] PTR BYTE INC    [BX] PTR BYTE DEC
ENCL2 JNZ    DX PUSH    DX PUSH    NEXT JMP
HERE LABEL ENCL3    DX PUSH    DX INC
DX PUSH    NEXT JMP    END-CODE
```

Fig: 8088/86:

CODE ENCLOSE

```
AX POP    BX POP    BX PUSH    0 AH, MOV
0FFF DX, MOV    BX DEC
HERE LABEL ENCL1    BX INC    DX INC    [BX] AH, CMP
ENCL1 JZ    DX PUSH    [BX] AH, CMP
ENCL2 JNZ    DX AX, MOV    DX INC    DPUSH JMP
HERE LABEL ENCL2    BX INC    DX INC    [BX] AL, CMP
ENCL4 JZ    [BX] AH, CMP    ENCL2 JNZ
HERE LABEL ENCL3    DX AX, MOV    DPUSH JMP
HERE LABEL ENCL4    DX AX, MOV    AX INC    DPUSH JMP
```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: WORD    BLK  @
  IF BLK  @  BLOCK ELSE TIB  @  THEN
  >IN  @  +  SWAP  ENCLOSE  DUP  4  PICK  -  FF  >
  IF WHAT  ." INPUT  >  255"  QUIT THEN
  HERE  22 BL  FILL  >IN  +!  OVER  -  >R  R@
  HERE  C!  +  HERE  1+  R>  CMOVE  HERE  ;
```

The example comes from an early version of the MVP-FORTH source code.

Comment: ENCLOSE is a highly specialized ideogram useful only for implementing WORD. In such usage, the ASCII null marks the end of the input stream and n3, the offset to the first character not included, is added directly into >IN.

END

NOT USED IN MVP-FORTH

A synonym for UNTIL. (*Fig*) This is an 'alias' or duplicate definition for UNTIL. (*79S Reference Word Set*) Same as MVP. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86: Could be defined as:

```
: END    [COMPILE]  UNTIL  ;    IMMEDIATE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: TEST    0
  BEGIN  1+ DUP  .  DUP  10  =  END  DROP  ;
```

A definition to print the numbers 1 through 10 ;

Comment: This ideogram is now obsolete and replaced by UNTIL. It may appear in older programs.

ENDCASE

END-CODE

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S Assembler Extension Word Set) Terminates a code definition and allows the <name> of the corresponding code definition to be found in the dictionary. sys is balanced with its corresponding CODE or ;CODE. See: CODE (F83) Terminates a code definition and restores vocs. (F-PC) Terminates CODE definitions.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

ASSEMBLER

: END-CODE AVOC @ CONTEXT ! REVEAL ;

F-PC:

: END-CODE

LL-GLOBAL? 0=

IF LL-ERRS? THEN

ARUNSAVE IS RUN PREVIOUS A; REVEAL ;

Example:

CODE NOOP NEXT JMP END-CODE

The example illustrates how a code definition is terminated.

Comment: All implementations of Forth have a function to accomplish this end. In some cases it is combined with an Assembler macro NEXT.

ENDCASE

NOT USED IN MVP-FORTH

```
addr1...addrn n --- (compiling)
n                --- (if no match)
                  --- (if match was found)
```

(Fig) Not defined. (79S) Not defined. (83S) Not defined (F83) Not defined. (F-PC) See CASE. (Eaker) Used in a colon definition in the form: CASE...OF...ENDOF...ENDCASE. Note that OF...ENDOF pairs may be repeated as necessary. At run-time, ENDCASE drops the select value if it does not equal any case values. ENDCASE then serves as the destination of forward branches from all previous ENDOF's. At compile-time ENDCASE compiles a DROP then computes forward branch offsets until all addresses left by previous ENDOF's have been resolved.

Finally, the value of CSP saved by CASE is restored. n is used for error checking.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

```
: ENDCASE
  CR SEQHANDLE COUNT TYPE 24 #OUT @ - SPACES
  LOADLINE @ 3 .R
  ." Warning ENDCASE has changed, "
  ." IT now does the DROP!"
  COMPILE DOENDCASE \ DOENDCASE does a DROP!!
  NRESOLVE DROP ; IMMEDIATE
```

Eaker:

```
: ENDCASE
  4 ?PAIRS COMPILE DROP
  BEGIN SP@ CSP @ = 0 =
  WHILE 2 [COMPILE] ENDIF
  REPEAT CSP ! ; IMMEDIATE
```

Example:

```
: GEN CASE
  0 OF IMMEDIATE ENDOF
  10 OF DIRECT ENDOF
  20 OF INDEXED ENDOF
  30 OF EXTENDED ENDOF
  MODE-ERROR
  ENDCASE RESET ;
```

The example is taken from the original article by Charles E. Eaker. The MODE-ERROR and RESET functions are taken from his implementation of fig-Forth and are not in common usage. They could be omitted from the example.

Comment: This is probably the most commonly used CASE function. It was published in Forth Dimensions Vol. II, No. 3, 1980 as one of three winners of FIG's CASE Statement Contest. All entries to the contest

ENDOF

were published in that issue.

ENDIF

NOT USED IN MVP-FORTH

addr n --- (compile)

(Fig) Occurs in a colon-definition. At run-time, ENDF serves only as the destination of a forward branch from IF or ELSE. It marks the conclusion of the conditional structure. THEN is another name for ENDF. Both names are supported in fig-Forth. See also IF and ELSE. At compile-time, ENDF computes the forward branch offset from addr to HERE and stores it at addr. n is used for error tests. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86: Could be defined as:

```
: ENDF [COMPILE] THEN ; IMMEDIATE
```

Fig: 8088/86:

```
: ENDF ?COMP 2 ?PAIR HERE OVER - SWAP ! ;  
IMMEDIATE
```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: TEST ?DUP IF . ENDF ;
```

With this new definition, if a value of 0 is on the stack, nothing will be done, otherwise, the value on the stack will be printed.

Comment: This ideogram is now obsolete and has been replaced by THEN. It may appear in older programs.

ENDOF

NOT USED IN MVP-FORTH

addr n1 --- addr2 n2 (compiling)

(Fig) Not defined. (79S) Not defined. (83S) Not defined (F83) Not defined. (F-PC) See CASE. (*Eaker*) Used in a colon definition in the form: CASE...OF...ENDOF...ENDCASE. Note that OF...ENDOF pairs may be repeated as necessary. At run-time, ENDOF transfers control to the code following the next ENDCASE provided there was a match at the last OF. If there was not a match at the last OF, ENDOF is the location to which execution will branch. At compile-time ENDOF

emplaces **BRANCH** reserving a branch offset, leaves the address **addr2** and **n2** for error checking. **ENDOF** also resolves the pending forward branch from **OF** by calculating the offset from **addr1** to **HERE** and storing it at **addr1**.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

: **ENDOF**

COMPILE DOENDOF >MARK SWAP >RESOLVE ; IMMEDIATE

Eaker:

: **ENDOF**

5 ?PAIRS COMPILE BRANCH HERE 3 ,
SWAP 2 [COMPILE] ENDIF 4 ; IMMEDIATE

Example:

: **GEN CASE**

0 OF IMMEDIATE ENDOF

10 OF DIRECT ENDOF

20 OF INDEXED ENDOF

30 OF EXTENDED ENDOF

MODE-ERROR

ENDCASE RESET ;

The example is taken from the original article by Charles E. Eaker. The **MODE-ERROR** and **RESET** functions are taken from his implementation of **fig-Forth** and are not in common usage. They could be omitted from the example.

Comment: This is probably the most commonly used **CASE** function. It was published in *Forth Dimensions* Vol. II, No. 3, 1980 as one of three winners of **FIG's CASE Statement Contest**. All entries to the contest were published in that issue.

ENTRY

--- A1

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Jum-

EPRINT

ped to during multitasking. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

VARIABLE ENTRY

F-PC:

Same as F83.

Example:

```
: TASK:
  CREATE TOS HERE #USER @ SMOVE
  HERE ENTRY LOCAL LINK !
  DUP HERE + DUP RPO ! 100 - SP0 ! SWAP UP !
  HERE #USER @ + HERE DP LOCAL !
  HERE SLEEP ALLOT ;
```

The example is a defining function as defined by Henry Laxen.

Note: Not all of the words are in common usage. Refer to Henry Laxen's Multi-Tasking, Part I and Part II in Forth Dimensions Vol. V, Nos 4 & 5, 1983/1984

Comment: Depending upon the application, multi-tasking can be accomplished in a number of ways.

EPRINT

--- addr

MVP-FORTH

A variable directing the output of Pemit through the system BIOS. 0 = Terminal Device; 1 = List Device. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

VARIABLE EPRINT 0 EPRINT !

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
1 EPRINT !
```

Change output of Pemit, and thereby also emit, to the system's list device.

Comment: This ideogram is used in most fig-Forth implementations, but is usually not available to the programmer. In MVP-FORTH, it is made available.

ERASE addr n --- MVP-FORTH SUPPLEMENTAL

Clear a region of memory to zero from addr over n addresses. (*Fig*) Same as MVP. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Fill the string with zeros. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

```
: ERASE 0 FILL ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Same as MVP.

F-PC:

Same as MVP.

Example:

```
PAD 40 ERASE
```

Beginning with the address of PAD, fill the next 40 bytes with nulls.

Comment: Though not present in all implementations of Forth it can easily be added with the FILL function.

ERROR line --- in blk NOT USED IN MVP-FORTH

(*Fig*) Execute error notification and restart of system. WARNING is first examined. If 1, the text of line n, relative to screen 4 of drive 0 is printed. This line number may be positive or negative, and beyond just screen 4. If WARNING is 0, n is just printed as a message number (non disk installation). If WARNING is -1, the definition (ABORT) is exe-

EXECUTE

cuted, which executes the system ABORT. The user may cautiously modify this execution by altering (ABORT). fig-Forth saves the contents of IN (now >IN) and BLK to assist in determining the location of the error. Final action is execution of QUIT. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

(This ideogram is not implemented.)

Fig: 8088/86:

HEX

```
: ERROR    WARNING @  0<
  IF (ABORT) ENDIF
  HERE COUNT TYPE ." ? " MESSAGE SP!
  ( CHANGE FROM FIG MODEL )
  >IN @ BLK @  2DUP
  IF >IN @ SWAP  ENDIF QUIT ;
```

F83: 8088/86:

Not defined.

F-PC:

Not defined.

Comment: Almost all implementations of Forth have a unique way of handling error messages. There are no standard techniques. Instead of using mysterious error numbers or messages out on disk, MVP-FORTH keeps memory resident error messages in-line with system code.

EXECUTE

addr ---

MVP-FORTH

Execute the dictionary entry whose compilation address is on the stack. (Fig) Execute the definition whose code field address is on the stack. The code field address is also called the compilation address. (79S) Same as MVP. (83S) The word definition indicated by addr is executed. An error condition exists if addr is not a compilation address. (F83) the word whose code field is on the stack. Very useful for passing executable routines to procedures!!! (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

CODE EXECUTE BX POP NEXT1 JMP END-CODE

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

CODE EXECUTE W POP 0 [W] JMP END-CODE

F-PC:

CODE EXECUTE POP AX JMP AX END-CODE

Example:

```
: CR 'CR @ EXECUTE ;
```

This example comes from the MVP-FORTH source code and illustrates vectoring.

Comment: This ideogram is most often used with execution vectors and with the code field addresses stored in variables. In some systems, EXECUTE takes a parameter field address instead of the code field address. Though there is only two bytes difference, it is sufficient to trigger catastrophe.

CAUTION — Trying to execute a garbage value may crash your system.

EXIT

MVP-FORTH

When compiled within a colon-definition, terminate execution of that definition, at that point. May not be used within a DO . . . LOOP. (*Fig*) Not defined. (79S) Same as MVP. (83S) Compile within a colon definition such that when executed, that colon definition returns control to the definition that passed control to it by returning control to the return point on the top of the return stack. An error condition exists if the top of the return stack does not contain a valid return point. May not be used within a do-loop. See: "stack, return" "9.3 Return Stack" (F83) Pop an entry off the return stack and place it into the Interpretive Pointer. Terminates a Hi Level definition. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

```
CODE EXIT
[BX] SI, MOV BP INC BP INC NEXT JMP END-CODE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
CODE EXIT SS: 0 [RP] IP MOV RP INC RP INC
NEXT
```

F-PC:

EXPECT

```
CODE EXIT
  XCHG RP, SP    \ 4 cycles
  POP IP         \ 8 cycles
  POP ES         \ 8 cycles
  XCHG RP, SP    \ 4 cycles
NEXT    END-CODE
```

Example:

```
: ;    COMPILE EXIT [COMPILE] [ ; IMMEDIATE
```

This is a simplified version of the ideogram “;”. No error checking or unsmudging is performed.

Comment: This ideogram replaces the now obsolete ;S in fig-Forth. When encountered outside of a colon definition, it will fool LOAD into thinking the end of the screen has been reached. This latter function will save some load time, but may not work on all systems.

EXPECT

addr n ---

MVP-FORTH

Transfer characters from the terminal beginning at addr, upward, until a “return” or the count of n has been received. Take no action for n less than or equal to zero. One or two nulls are added at the end of the text. (*Fig*) Transfer characters from the terminal to address, until a “return” or the count of characters have been received. One or more nulls are added at the end of the text. (*79S*) Same as MVP. (*83S*) Receive characters and store each into memory. The transfer begins at addr proceeding towards higher addresses one byte per character until either a “return” is received or until +n characters have been transferred. No more than +n characters will be stored. The “return” is not stored into memory. No characters are received or transferred if +n is zero. All characters actually received and stored into memory will be displayed, with the “return” displayed as a space. See: SPAN “9.5.2 EXPECT” (*F83*) Get a string from the terminal and place it in the buffer provided. Performs a certain amount of line editing. Saves the number of characters input in the Variable SPAN. Processes control characters per the array pointed to by CC. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

```
: EXPECT    'EXPECT    @    EXECUTE    ;
```

Fig: 8088/86:

```
HEX
CODE EXPECT
```

```

OVER + OVER
DO KEY DUP 0E +ORIGIN @ =
  IF DROP DUP I = DUP R> 2 - + >R
    IF 07 ELSE 08 ENDIF
  ELSE DUP 0D =
    IF LEAVE DROP BL 0 ELSE DUP ENDIF
    I C! 0 I 1+ !
  ENDIF EMIT
LOOP DROP ;

```

F83: 8088/86:

```

: EXPECT
  DUP SPAN ! SWAP 0
  BEGIN 2 PICK OVER - ( len adr #so-far #left )
  WHILE KEY DUP LAST-KEY ! DUP BL <
    IF 2* CONTROL-CHARS @ + PERFORM
    ELSE 127 = IF DEL-IN ELSE CHAR THEN
  THEN
  REPEAT 2DROP DROP ;

```

F-PC:

```

: (EXPECT) 0 NEXPECT ;
DEFER EXPECT ' (EXPECT) IS EXPECT

```

Example:

```

: QUERY TIB @ 50 EXPECT 0 >IN ! ;

```

This example comes from the MVP-FORTH implementation.

Comment: This ideogram takes a line from the input terminal and places it anywhere you like. EXPECT includes some features, such as backspace handling, which may be installation dependent. Therefore, in later versions of MVP- FORTH, it's run-time implementation has been vectored through ' EXPECT.

FALSE

--- f1

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Constants for clarity. (F-PC) Constants for clarity.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

FENCE

F83: 8088/86:

0 CONSTANT FALSE

F-PC:

Same as F83.

Example:

FALSE

Place a 0 on the stack.

Comment: Gives a name to the value for false. It serves as a parallel with TRUE and the actual values can be changed without going through all of the program.

FENCE

--- addr

MVP-FORTH

A user variable containing an address below which FORGETTING is trapped. To forget below this point the user must alter the contents of FENCE. (Fig) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Limit address for forgetting. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

HEX 10 USER FENCE

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

VARIABLE FENCE

F-PC:

Same as F83.

Example:

' LIST NFA FENCE !

Change the protected part of the dictionary to the name field address of LIST. Subsequently, it will be possible to forget from the top of the dictionary back to and including LIST.

Comment: This variable is set by a value in low memory at the time of system boot or execution of COLD. By changing its value the boundary between the full Forth implementation and the application definitions can be changed.

FILL addr n byte --- MVP-FORTH

Fill memory beginning at address with a sequence of n copies of byte. If the quantity is less than or equal to zero, take no action. (*Fig*) Fill memory at the address with the specified quantity of bytes b. (*79S*) Same as MVP. (*83S*) u bytes of memory beginning at addr are set to 8b. No action is taken if u is zero. (*F83*) FILL the string starting at start-adr for count bytes with the character char. Both BLANK and ERASE are special cases of FILL. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

```
: FILL OVER 0>
  IF <FILL>
    ELSE DDROP DROP THEN ;
```

Fig: 8088/86:

```
CODE FILL
  AX POP CX POP DI POP DS BX, MOV
  BX ES, MOV CLD REP AL STOS NEXT JMP
```

F83: 8088/86:

```
CODE FILL
  CLD DS AX MOV AX ES MOV AX POP
  CX POP DI POP REP AL STOS NEXT
```

F-PC:

```
CODE FILL
  CLD MOV BX, DS POP AX POP CX POP DI
  PUSH ES MOV ES, BX REPNZ STOSB POP ES
  NEXT END-CODE
```

Example:

```
: BLANK BL FILL ;
```

This example is taken from the MVP-FORTH implementation.

Comment: ERASE and BLANKS, if not already in the system, may easily be defined using this ideogram.

FIND --- addr MVP-FORTH

Leave the compilation address of the next word name, which is accepted from the input stream. If that word cannot be found in the dictionary after a search of CONTEXT and FORTH leave zero. (*Fig*) Not defined. (*79S*) Same as MVP. (*83S*) addr1 is the address of a counted

FIND

string. The string contains a word name to be located in the currently active search order. If the word is not found, addr2 is the string address addr1, and n is zero. If the word is found, addr2 is the compilation address and n is set to one of two non-zero values. If the word found has the immediate attribute, n is set to one. If the word is non-immediate, n is set to minus one (true). (F83) Run through the vocabulary list searching for the name whose address is supplied on the stack. If the name is found, return the code field address of the name and a non-zero flag. The flag is -1 if the word is non-immediate and 1 if it is immediate. If the name is not found, the string address is returned along with a false flag. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

```
: FIND  -FIND  IF  DROP  CFA  ELSE  0  THEN  ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: FIND  DUP C@
  IF  FALSE  #VOCS 0
    DO  DROP CONTEXT I 2* + @ DUP
      IF  OVER 1+ @ SWAP HASH @ (FIND)  DUP ?LEAVE  THEN
        LOOP
      ELSE  DROP  ['] NOOP 1
    THEN  ;
```

F-PC:

```
: %FIND
  MOV DI, SP  MOV BX, 0 [DI]
  CMP 0 [BX], # 0 BYTE 0<>
  IF  MOV PRIOR # 0 WORD  MOV BX, #0  PUSH BX
    MOV CX, # #VOCS  PUSH CX  PUSH BX
    MOV AX, # ' %%FIND  JMP AX
  THEN  MOV END? # TRUE WORD
  MOV 0 [DI], # ' NOOP WORD  MOV AX, # 1
  1PUSH  END-CODE
  DEFER FIND  ' %FIND IS FIND
```

Example:

```
FIND DUP
```

Leaves the code field address of the ideogram, DUP, on the stack.

Comment: This ideogram is closely related to the older -FIND. Note

that **FIND** leaves the code field address while **-FIND** leaves the parameter field address and a length byte.

FIRST --- n MVP-FORTH

A constant that leaves the address of the first (lowest) block buffer. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) This is a simple constant having the value 10 less than **LIMIT**.

Implementations:

MVP: 8088/86:
 BUF1 CONSTANT FIRST

Fig: 8088/86:
 Same as MVP.

F83: 8088/86:
 LIMIT B/BUF #BUFFERS * - CONSTANT FIRST

F-PC:
 LIMIT 10 - CONSTANT FIRST

Example:

FIRST

This ideogram is a constant which leaves the address on the stack.

Comment: The value of this constant is implementation dependent. It is not available in all implementations of Forth. In MVP-FORTH, its value may be modified dynamically with **CHANGE**.

FLD --- addr MVP-FORTH

A variable pointing to the field length reserved for a number during output conversion. (*Fig*) A user variable for control of number output field width. Presently unused in fig-Forth. (79S *Reference Word Set*) Same as MVP. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:
 HEX 44 USER FLD

Fig: 8088/86:
 HEX 2A USER FLD

FLUSH

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

FLD @ .

Fetch the value of this variable and print it.

Comment: Though defined in the *Installation Manual*, it has not been implemented in fig-Forth or 79S but is included in the 79S Reference Word Set.

FLIP n1 --- n2 NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Exchange the hi and low halves of a word. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE FLIP AX POP AH AL XCHG 1PUSH

F-PC:

Same as F83.

Example:

HEX 1234 FLIP U. DECIMAL

The example will exchange the first two bytes with the second and return the value 3412.

Comment: The function facilitates packing byte values into 16-bit integer space.

FLUSH --- MVP-FORTH SUPPLEMENTAL

A synonym for SAVE-BUFFERS. (Fig) Not defined. (79S Reference Word Set) Same as MVP. (83S) Perform the function of SAVE-BUFFERS then unassign all block buffers. (This may be useful for mounting or changing mass storage media). (F83) Save and empties the buffers.

Used for changing disks. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: FLUSH    SAVE-BUFFERS    ;
```

Fig: 8088/86:

```
: FLUSH    #BUF 1+ 0  
    DO 0 BUFFER DROP LOOP    ;
```

F83: 8088/86:

```
: FLUSH    SAVE-BUFFERS    EMPTY-BUFFERS    ;
```

F-PC:

Not used.

Example:

FLUSH

This ideogram, without taking any parameters, causes all buffers marked for UPDATE to be written back to disk.

Comment: This obsolete synonym for SAVE-BUFFERS may be included for compatibility with older programs. However, its use is well established and it is included in the 79S Reference Word Set. FLUSH may therefore continue to be used. It is included in MVP-FORTH.

FORGET

MVP-FORTH

Delete from the dictionary <name> (which is in the CURRENT vocabulary) and all words added to the dictionary after <name> regardless of their vocabulary. Failure to find <name> in CURRENT or FORTH is an error condition. (*Fig*) Executed in the form: FORGET cccc. Deletes definition named cccc from the dictionary with all entries physically following it. In fig-Forth an error message will occur if the CURRENT and CONTEXT vocabularies are not currently the same. (79S) Same as MVP. (83S) Used in the form: FORGET <name>. If <name> is found in the compilation vocabulary, delete <name> from the dictionary and all words added to the dictionary after <name> regardless of their vocabulary. Failure to find <name> is an error condition. An error condition also exists if the compilation vocabulary is deleted. See: "10.2 General Error Conditions". (F83) Forget all of the code and headers before the next word. (*F-PC*) Forget all of the code and headers before <name>. FORGET must be used with caution, if you are using DEFERS or UDEFERS, FORGET will not know about these chains, so they must be unlinked before using FORGET or disaster WILL strike.

FORGET

Form: FORGET <name>

Implementations:

MVP: 8088/86:

```
: FORGET BL WORD CURRENT @ @ <FIND> 0=
  ABORT" NOT IN CURRENT VOCABULARY "
  DROP NFA DUP FENCE @ U<
  ABORT" IN PROTECTED DICTIONARY "
  >R R@ CONTEXT @ U<
  IF [COMPILE] FORTH THEN
  R@ CURRENT @ U<
  IF [COMPILE] FORTH DEFINITIONS THEN
  VOC-LINK @
  BEGIN R@ OVER U< WHILE @ REPEAT
  DUP VOC-LINK !
  BEGIN DUP 4 -
    BEGIN PFA LFA @ DUP R@ U< UNTIL
    OVER 2- ! @ ?DUP 0=
  UNTIL R> DP ! ;
```

Fig: 8088/86:

HEX

```
: FORGET CURRENT @ CONTEXT @ - 18 ?ERROR
  ' DUP FENCE @ < 15 ?ERROR DUP
  NFA DP ! LFA @ CONTEXT @ ! ;
```

F83: 8088/86:

```
: (FORGET)
  DUP FENCE @ U< ABORT" Below fence"
  VOC-LINK @
  BEGIN 2DUP U<
  WHILE @
  REPEAT
  DUP VOC-LINK ! OVER >LINK SWAP
  BEGIN DUP
  WHILE 2DUP #THREADS 2* - TRIM @
  REPEAT DROP DP-H ! DP ! ;
: FORGET
  TOKEN DUP 1+ @ CURRENT @ HASH @ (FIND) 0= ?MISSING
  (FORGET) ;
```

F-PC:

```
: (FRGET)
  DUP FENCE @ U< ABORT" Below fence" ( ca va )
```

```

OVER VOC-LINK @
BEGIN 2DUP U< WHILE @ REPEAT
DUP VOC-LINK ! ( ca va ca pt ) NIP
BEGIN DUP
WHILE 2DUP #THREADS 2* - TRIM @
REPEAT
DROP YDP ! DUP 1+ @ OVER >BODY + (LIT)
TRIM DUP 1+ @ SWAP >BODY + = \ If it's a : def
IF DUP >BODY @ +XSEG XDPSEG !
\ Set back XHERE too!
XDP OFF
THEN DP ! ;
: FORGET
BL WORD ?UPPERCASE DUP CURRENT @ HASH @
(FIND) 0= ?MISSING DUP >VIEW (FRGET) ;

```

Example:

```
FORGET TEST
```

This ideogram will forget everything from the top of the dictionary through the most recent definition of TEST.

Comment: This is an implementation of a smart FORGET as discussed in *Forth Dimensions*, Vol II, No. 6. With multiple vocabularies containing ideograms added at various times, all ends must be properly linked together or the system will crash. This implementation, though appearing moderately complex, is quite safe. FENCE serves as a movable address boundary to protect ideograms defined prior to that point. Ragsdale's vocabulary modifications will modify the implementation.

FORTH

MVP-FORTH

The name of the primary vocabulary. Execution makes FORTH the CONTEXT vocabulary. New definitions become a part of Forth until a differing CURRENT vocabulary is established. User vocabularies conclude by chaining to FORTH, so it should be considered that FORTH is 'contained' within each users' vocabulary. (Fig) The name of the primary vocabulary. Execution makes FORTH the CONTEXT vocabulary. Until additional user vocabularies are defined, new user definitions become a part of FORTH. FORTH is immediate, so it will execute during the creation of a colon-definition, to select this vocabulary at compile time. (79S) Same as MVP. (83S) The name of the primary vocabulary. Execution replaces the first vocabulary in the search order with FORTH. FORTH is initially the compilation vocabulary and the first vocabulary

FORTH-83

in the search order. New definitions become part of the FORTH vocabulary until a different compilation vocabulary is established. See: VOCABULARY (F83) Not defined. (F-PC) - The Forth vocabulary, where most user words can be found. All of the words in this glossary are in the FORTH vocabulary.

Implementations:

MVP: 8088/86:

VOCABULARY FORTH IMMEDIATE

Fig: 8088/86:

FORTH DOES DOVOC , 0A081 , TASK-7 , 0 , ;

F83: 8088/86:

VOCABULARY FORTH

F-PC:

Same as F83.

Example:

FORTH

Set CONTEXT to point to FORTH and leave CURRENT unchanged.

Comment: MVP-FORTH conforms with 79S. All vocabularies chain only to FORTH. In fig-Forth, daughter vocabularies chain to their parents before Forth. There is some ambiguity in the definition of the proper function and not all implementations of Forth are the same. Ragsdale's vocabulary modification will modify the implementation.

FORTH-83

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Assures that a FORTH-83 Standard System is available, otherwise an error condition exists. (F83) Let's hope so. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

: FORTH-83 FORTH DEFINITIONS CAPS OFF ;

F-PC:

Not used.

Example:

FORTH-83

The example appears to do nothing and returns the usual prompt.

Comment: It is unclear why an implementation would return an error condition.

FREEZE

MVP-FORTH

Save the current values of the user variables and the top of the dictionary in low memory in place of the original values. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: FREEZE  UP  @  6  +  INIT-USER  30  CMOVE
  '  FORTH  2+  @  INIT-FORTH  !  ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

FREEZE

This ideogram requires no parameters. It sets the start-up values in low memory to the current values of the user variables. When COLD is executed later, the system will return to the configuration at the time of the most recent execution of FREEZE. Nothing is written to disk.

Comment: This ideogram reconfigures low memory, changing the startup parameters. A new startup image can be saved on disk.

GO

addr ---

MVP-FORTH

Makes the address on the stack the next address in the hardware program counter. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Execute code at the given address. (*F-PC*) Same as F83.

H

Implementations:

MVP: 8088/86:

Used in 8080 implementation.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE GO RET END-CODE

F-PC:

Same as F83.

Example:

HEX 100 GO

Execution of this example is equivalent to COLD.

Comment: The implementation of this ideogram is system dependent. As used in the implementation of BYE, it will return to the operating system.

H

--- addr

MVP-FORTH SUPPLEMENTAL

A synonym for DP, the dictionary pointer. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) (An alias for HELP.)

Implementations:

MVP: 8088/86:

: H DP ;

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

' HELP ALIAS H

Example:

H @ U.

This ideogram is identical to DP and returns the address of the value returned by HERE which in this case is then printed.

Comment: Though not used in the general Forth community, this ideogram has been in use for a number of years in the University of

Return the address of the next available dictionary location. (*Fig*)
 Leave the address of the next available dictionary location. (*79S*)
 Same as MVP. (*83S*) The address of the next available dictionary
 location. (*F83*) Return the address of the top of the dictionary (*F-PC*)
 Same as F83.

HIDE

HEX

: HEX 10 BASE ! ;

Fig: 8088/86:

: HEX 16 BASE ! ;

F83: 8088/86:

: HEX 16 BASE ! ;

F-PC:

Same as F83.

Example:

HEX FF DECIMAL .

Convert the hex value FF to decimal and print it: 255 .

Comment: This ideogram is not included in 79S. It may be easily defined if you know the current value of BASE.

HIDE

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Removes the Last definition from the Header Dictionary. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

: HIDE

LAST @ DUP N>LINK H@ SWAP 1+ H@ CURRENT @ HASH ! ;

F-PC:

: HIDE

LAST @ DUP N>LINK Y@ SWAP CURRENT @ YHASH ! ;

Example:

: TEST 10 1 DO I U. LOOP ; TEST HIDE TEST

A contrived example. Define a function, TEST, execute it and then HIDE it from a search of the dictionary. The second execution does not find the function

Comment: Used to remove words not needed in searches of the dictio-

nary.

HLD --- addr

MVP-FORTH

A user variable that holds the address of the latest character of text during numeric output conversion. (*Fig*) Same as MVP. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Points to a converted character during numeric output. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

HEX

46 USER HLD

Fig: 8088/86:

HEX

30 USER HLD

F83: 8088/86:

VARIABLE HLD

F-PC:

Same as F83.

Example:

HLD @ U.

Get the value of the user variable HLD, and print it unsigned.

Comment: In fig-Forth and MVP-FORTH, this user variable determines the number of characters below PAD which contain the formatted string resulting from a binary number conversion. Not all implementations utilize this method of formatting number conversions.

HOLD char ---

MVP-FORTH

Insert char into a pictured numeric output string. May only be used between <# and #>. (*Fig*) Used between <# and #> to insert an ASCII character into a pictured numeric output string. e.g. 2E HOLD will place a decimal point. (*79S*) Same as MVP. (*83S*) char is inserted into a pictured numeric output string. Typically used between <# and #> . (*F83*) Save the char for numeric output later. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

: HOLD -1 HLD +! HLD @ C! ;

HPUSH

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Same as MVP.

F-PC:

: HOLD HLD DECR HLD @ C! ;

Example:

3.33 <# # # 46 HOLD #S #> TYPE

Enter the double precision value with two digits to the right of the decimal and then print it in the same format.

Comment: The ASCII value for the desired character is used to form the output picture such as decimal 47 for each slash when outputting the date as 01/01/81. Note the backward order of the picture generation.

HPUSH

--- addr

MVP-FORTH

A constant used in 8080/Z80 implementations pointing to a machine code entry point which pushes the contents of the HL register onto the stack and then falls through to NEXT, the inner interpreter. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

CODE +

D POP H POP D DAD HPUSH JMP END-CODE

This example is taken from the MVP-FORTH 8080 source code.

Comment: This is a machine, system, and implementation dependent

location used only at the machine code level. It allows trimming one byte off CODE definitions which would otherwise end with H PUSH NEXT JMP.

I --- n MVP-FORTH

Copy the loop index onto the data stack. May be only used in the DO-LOOP control structure. (*Fig*) Used within a DO-LOOP to copy the loop index to the stack. Other use is implementation dependent. See R. (79S) Same as MVP. (83S) w [n above] is a copy of the loop index. May only be used in the form [shown below]. (F83) returns the current loop index. It now requires a little more calculation to compute it than in Fig Forth but the tradeoff is a much faster (LOOP). The loop index is stored on the Return Stack. (F-PC) Same as F83.

Form: DO ... I ... LOOP (or +LOOP)

Implementations:

MVP: 8088/86:

```
CODE I [BP] AX, MOV APUSH JMP END-CODE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE I
SS: 0 [RP] AX MOV SS: 2 [RP] AX ADD 1PUSH
```

F-PC:

```
CODE I
MOV AX, 0 [RP] ADD AX, 2 [RP] 1PUSH END-CODE
```

Example:

```
: TEST 10 0 DO I . LOOP ;
```

This new definition will print the values of the digits 0 through 9.

Comment: In this implementation the indices and limits are held on the return stack. Thus, I is synonymous with R@ . Some implementations hold the loop parameters in a separate stack.

I' --- n MVP-FORTH

Used within a colon-definition executed only from within a DO-LOOP to return the corresponding loop index. (*Fig*) Not defined. (79S *Reference Word Set*) Same as MVP. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

ID.

Pronounced: i-prime

Implementations:

MVP: 8088/86:

```
CODE I' [BX] 2+ AX, MOV APUSH JMP END-CODE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: TEST 10 0
DO 45 >R I' . R> DROP LOOP ;
```

This contrived example will print the digits from 0 through 9.

Comment: This ideogram is useful for accessing the index when it has been buried one level in the return stack. The return stack must be restored at the end of the DO-LOOP. This ideogram is not available in all implementations.

ID. addr ---

MVP-FORTH UTILITY

Print a definition's name from its name field address. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: ID.
COUNT 1F AND OVER SWAP
DO I .INDEX PAUSE ?TERMINAL
IF LEAVE THEN 1
/LOOP TYPE ;
```

Fig: 8088/86:

HEX

```
: ID.
PAD 20 5F FILL DUP PFA LFA OVER - PAD SWAP CMOVE
PAD COUNT 1F AND TYPE SPACE ;
```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
' ID. @ 2+ NFA ID.
```

This example could be used as a factor in a decompiler. Since `ID.` is a colon definition, its parameter field address contains the code field address of its first ideogram. The `2+` moves to its parameter field address and then `NFA` to its name field address. Then `ID.` will print the first ideogram in its definition: `COUNT`.

Comment: Such poking around the header structure is prohibited in the 79S. However, it can be most useful in a development system. Clearly, its implementation depends on the structure of the header. If the ideogram being decompiled is not a colon definition, `ID.` as used in the example, will produce garbage. For names truncated to less than their natural `WIDTH` when defined, `ID.` will represent the lost character(s) with garbage. Other dialects of Forth use `.ID.`

IF `flag ---`

MVP-FORTH

Used only in a colon definition. If `flag` is true, the words following `IF` are executed and the words following `ELSE` are skipped. The `ELSE` part is optional. If `flag` is false, words between `IF` and `ELSE`, or between `IF` and `THEN` (when no `ELSE` is used), are skipped. `IF-ELSE-THEN` conditionals may be nested. (*Fig*) Occurs in a colon-definition ... At run-time, `IF` selects execution based on a boolean flag. If `f` is true (non-zero), execution continues ahead through the true part. If `f` is false (zero), execution skips till just after `ELSE` to execute the false part. After either part, execution resumes after `ENDIF`. `ELSE` and its false part are optional; if missing, false execution skips to just after `ENDIF`. At compile-time `IF` compiles `0BRANCH` and reserves space for an offset at `addr`. `addr` and `n` are used later for resolution of the offset and error testing. (79S) Same as MVP. (83S) Used in the form ... If `flag` is true, the words following `IF` are executed and the words following `ELSE` until just after `THEN` are skipped. The `ELSE` part is optional. If `flag` is false, words from `IF` through `ELSE`, or from `IF` through `THEN` (when no `ELSE` is used), are skipped. `sys` is balanced with its corresponding `ELSE` or `THEN`. See: "9.9 Control Structures". (*F83*) Not defined. (*F-PC*) Used in the form: `f1 IF <true> ELSE <false> THEN` (`ELSE` is optional). If flag `f1` is false, branches forward to `<false>` or after `THEN`.

IMMEDIATE

Form: IF ... ELSE ... THEN or
IF ... THEN

Implementations:

MVP: 8088/86:

```
: IF    COMPILE  OBRANCH  HERE  0  ,  2  ;  
IMMEDIATE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
: IF    COMPILE  ?BRANCH  >MARK  1000  ; IMMEDIATE
```

F-PC:

```
: IF    COMPILE  ?BRANCH  ?>MARK    ; IMMEDIATE
```

Example:

```
: TEST  1  IF  ."  ONE  "  THEN  ;
```

This definition will always print the text "ONE".

Comment: An indispensable control structure. However, in place of extensive nesting, it may be more efficient to define and use one of the several CASE utilities which have been defined in *Forth Dimensions*.

IMMEDIATE

MVP-FORTH

Mark the most recently made dictionary entry as a word which will be executed when encountered during compilation rather than compiled. (Fig) Mark the most recently made definition so that when encountered at compile time, it will be executed rather than being compiled. i.e. the precedence bit in its header is set. This method allows definitions to handle unusual compiling situations, rather than build them into the fundamental compiler. The user may force compilation of an immediate definition by preceeding it with [COMPILE]. (79S) Same as MVP. (83S) Marks the most recently created dictionary entry as a word which will be executed when encountered during compilation rather than compiled. (F83) Mark the last Header as an Immediate word. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

```
HEX  
: IMMEDIATE  LATEST  40  TOGGLE  ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
: IMMEDIATE \ 64 ( Precedence bit ) LAST @ CSET
;
LAST @ 64 OVER H@ OR SWAP H! ;
```

F-PC:

```
: IMMEDIATE 64 ( Precedence bit ) LAST @ YCSET ;
```

Example:

```
: ( -1 >IN +! 29 WORD DROP ; IMMEDIATE
```

This example comes from the MVP-FORTH implementation.

Comment: This ideogram forces execution during compiling by flipping the precedence bit in the most recently defined name. When the precedence bit is set, the ideogram will execute, regardless of whether the system is compiling or executing.

IN

--- addr

NOT USED IN MVP-FORTH

(Fig) A user variable containing the byte offset within the current input text buffer (terminal or disk) from which the next text will be accepted. WORD uses and moves the value of IN. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Assembler function only.

Implementations:

MVP: 8088/86: Could be defined as:

```
: IN >IN ;
```

Fig: 8088/86:

```
HEX 18 USER IN
```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
IN @ .
```

Get the present value of the offset in the input stream and print it.

Comment: This ideogram is now obsolete having been replaced by >IN in 79S. It may appear in older programs.

INDEX

INDEX from to --- MVP-FORTH UTILITY

Print the first line of each screen over the range: from, to. This is used to view the comment lines of an area of text on disk screens. (*Fig*) Same as MVP. (*79S Reference Word Set*) Print the first line of each screen over the range {n1..n2}. This displays the first line of each screen of source text, which conventionally contains a title. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Display an index of the first line of each file that matches the file_specs included on the line following INDEX.

Implementations:

MVP: 8088/86:

```
: INDEX
  CR OVER BLOCK DROP 1+ BPDRV MIN SWAP BPDRV MIN
  DO I .INDEX PAUSE ?TERMINAL IF LEAVE THEN 1 /LOOP
;
```

Fig: 8088/86:

HEX

```
: INDEX 00FF EMIT CR 1+ SWAP
DO CR I 3 .R SPACE 0 I .LINE
?TERMINAL IF LEAVE ENDIF
LOOP ;
```

F83: 8088/86:

```
: INDEX
  2 ?ENOUGH 1+ SWAP
DO I .LINE0
LOOP CR ;
```

F-PC:

```
: INDEX
." \n\n**** Use SPACE to pause, and EXC to
stop. ****\n\:03"
WITHNAME ON ?IN-EMPTY \ if nothing following command
IF " * *.SEQ" ">$>TIB \ substitute "*.seq"
  WITHNAME OFF
THEN ['] .FIRSTLINE FALLOF
RESTORESTATE ;
```

Example:

```
20 40 INDEX
```

This will cause the first line of each of the screens 20 through 40, inclusive, to be printed in succession.

Comment: If you adopt the convention of putting a descriptive comment on line 0 of every screen, INDEX will generate something like a table of contents. This MVP-FORTH version incorporates a PAUSE feature, which holds the display still when any key is pressed. Once suspended, the INDEX may be resumed by striking any key once, or aborted by striking any two keys in rapid succession.

F-PC does a similar listing of first lines from files. Thus the first lines should be descriptive.

INIT-FORTH

--- addr

MVP-FORTH

A constant locating the bootup parameter used to initialize the Forth vocabulary. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

INIT-FORTH CONSTANT INIT-FORTH

(The cross-compiler uses the label INIT-FORTH to define the constant INIT-FORTH).

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

INIT-FORTH @ ID.

Fetch the name field address of the topmost ideogram in the current cold-start image. This address is passed, without adjustment, to ID. which types the name at the terminal.

Comment: This element within the INIT-USER array is important enough to deserve its own ideogram. A COLD start stores this value into the body of the FORTH vocabulary. Executing FREEZE reverses the process, expanding or cutting back the bootup system image to match the current system configuration.

INIT-USER

--- addr

MVP-FORTH

A constant returning a pointer to the start of the bootup parameter

INTCALL

area in low memory. This area is an array containing cold-start values for the user variables. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
INIT-USER  CONSTANT  INIT-USER
```

(The cross-compiler uses the label INIT-USER to define the constant INIT-USER).

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
INIT-USER  U.
```

Print the beginning address of the low memory array containing the initial user variables.

Comment: Access to the array allows one to modify the initial values of some of the user variables, as is done in FREEZE and CHANGE.

INTCALL

ax bx cx dx int# --- n

MVP-FORTH

Pass the parameters to set up the respective registers and the interrupt number to the ideogram so the 8086/8088 system interrupt calls can be made directly from high-level Forth. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: INTCALL [ INTCA2 1 + ] LITERAL C!
```

```
INTCA1 EXECUTE ;
```

```
ASSEMBLER
```

```
HERE LABEL INTCA1 INTCA1 2+ ,
```

```
DX POP CX POP BX POP AX POP
```

```
SI PUSH BP PUSH
```

```
HERE LABEL INTCA2 0 INT
```

```
BP POP SI POP APUSH JMP
```

```
( This is a sneaky bit of implementation. )
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: Pemit ( b --- )  
0 0 0 23 intcall ;
```

Output the ASCII character in byte *b* to the printer on the IBM Personal Computer via an interrupt call.

Comment: The implementation of this ideogram is system dependent. It has been added to the vocabulary for the convenience of IBM users. For example, with it, new disk drivers can be written in high-level Forth and vectored to the proper functions.

INTERPRET

MVP-FORTH

Begin interpretation at the character indexed by the contents of >IN relative to the block number contained in BLK, continuing until the input stream is exhausted. If BLK contains zero, interpret characters from the terminal input buffer. (*Fig*) The outer text interpreter which sequentially executes or compiles text from the input stream (terminal or disc) depending on STATE. If the word name cannot be found after a search of CONTEXT and then CURRENT it is converted to a number according to the current base. That also failing, an error message echoing the name with a "?" will be given. Text input will be taken according to the convention for WORD. If a decimal point is found as part of a number a double number value will be left. The decimal point has no other purpose than to force this action. See NUMBER. (*79S Reference Word Set*) Same as MVP. (*83S*) Not defined. (*F83*) The Forth Interpret Loop. If the next word is defined, execute it, otherwise convert it to a number and push it onto the stack. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

```
: INTERPRET 'INTERPRET @ EXECUTE ;
```

Fig: 8088/86:

HEX

```
; INTERPRET -FIND
```

```
BEGIN
  IF STATE @ <
    IF CFA , ELSE CFA EXECUTE ENDIF ?STACK
    ELSE HERE NUMBER DPL @ 1+
    IF DLITERAL ELSE DROP LITERAL ENDIF ?STACK
  ENDIF
  AGAIN ;
```

F83: 8088/86:

```
: INTERPRET
  BEGIN ?STACK TOKEN DUP C@
  WHILE EVAL
  REPEAT DROP ;
```

F-PC:

```
: INTERP
  BEGIN ?STACK DEFINED
  IF EXECUTE
  ELSE NUMBER DOUBLE? NOT IF DROP THEN
  THEN FALSE DONE?
  UNTIL ;
  DEFER INTERPRET ' INTERP IS INTERPRET
```

Example:

```
: AD BLK @ >R >IN @ >R 0 >IN !
  BLK ! INTERPRET R> >IN ! R> BLK ! ;
```

This example is taken from the MVP-FORTH source code.

Comment: The ideogram used to interpret text source in MVP-FORTH. This ideogram is vectored in MVP-FORTH for the convenience of the programmer. Normally, it calls <INTERPRET>.

IS cfa --- NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Depending on STATE, either sets the following DEFERred word immediately or compiles the setting for later. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: IS
  STATE @ IF COMPILE (IS) ELSE ' >IS ! THEN ;
  IMMEDIATE
```

F-PC:

Same as F83.

Example:

```
DEFER KEY? ' (KEY?) IS KEY?
```

The example is taken from the F83 source code. The function of KEY? is vectored to the primitive (KEY?).

Comment: This is an excellent way to vector functions which you may want to change. C. H. Moore has used this ideogram in conjunction with a defining work like VALUE as Paul Bartholdi used TO.

J

--- n

MVP-FORTH

Return the index of the next outer loop. May be used only within a nested DO- LOOP. (*Fig*) Not defined. (79S) Same as MVP. (83S) w is a copy of the index of the next outer loop. May only be used within a nested DO-LOOP or DO-+LOOP in the form: ... (*F83*) returns the loop index of the inner loop in nested DO ... LOOPS. (*F-PC*) Same as F83.

Form: DO ... DO ... J ... LOOP ... LOOP (or +LOOP)

Implementations:

MVP: 8088/86:

```
CODE J [BP] 4 + AX, MOV APUSH JMP END-CODE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
CODE J
  SS: 6 [RP] AX MOV SS: 8 [RP] AX ADD 1PUSH
  DECIMAL
```

F-PC:

```
CODE J MOV AX, 6 [RP] ADD AX, 8 [RP]
  1PUSH END-CODE
```

Example:

```
: TEST 3 0 DO CR 10 0
  DO J . LOOP
```

KEY

```
LOOP ;
```

This definition will print a row of 10 0's, 10 1's, and 10 2's.

Comment: This utility allows reference to both loops from within the inner one as may be required for some two dimensional array applications.

KEY

--- char

MVP-FORTH

Leave the ASCII value of the next available character from the current input device. (*Fig*) Leave the ASCII value of the next terminal key struck. (79S) Same as MVP. (83S) The least-significant 7 bits of 16b is the next ASCII character received. All valid ASCII characters can be received. Control characters are not processed by the system for any editing purpose. Characters received by KEY will not be displayed. See: "9.5.1 KEY" (F83) Usually set to (KEY) to get a character from the user. (F-PC) A deferred word to get a key from user.

Implementations:

MVP: 8088/86:

```
: KEY 'KEY @ EXECUTE ;
```

Fig: 8088/86:

```
CODE KEY PKEY JMP
```

F83: 8088/86:

```
DEFER KEY
```

```
: (KEY) 0 0 BDOS ;
```

```
(KEY) IS KEY
```

F-PC:

```
DEFER KEY
```

```
: (KEY)
```

```
  BEGIN PAUSE KEY? UNTIL
```

```
  BIOSKEY DUP 127 AND 0=
```

```
  IF FLIP 3 =
```

```
    IF DROP 0 \allow a NULL
```

```
    ELSE 127 AND 128 OR
```

```
    THEN
```

```
  ELSE 255 AND
```

```
  THEN KEYFILTER ;
```

```
' (KEY) IS KEY ( NOTE: Depends upon system )
```


*Example:***KEY**

Execution of this ideogram causes the program to wait for any single input from the keyboard and upon receiving it places the ASCII value of the input on the stack.

Comment: The ideogram provides a way of finding out the ASCII value of characters without reference to a chart. It may also be used in selecting from a menu requiring only a single character input or for a wait until any character is input from the terminal. The internal details of KEY are installation dependent. Therefore, KEY has been vectored in MVP-FORTH, defaulting to <KEY>.

KEY?

--- f1

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Usually set to (KEY?), to sense keyboard status. (F-PC) A deferred word that returns a true flag if a key waiting.

*Implementations:***MVP: 8088/86:**

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

DEFER KEY?

: (KEY? 0 11 BDOS 0<> ;

' (KEY?) IS KEY?

F-PC:

DEFER KEY?

' MOUSEKEY? IS KEY?

Example:

' (KEY?) IS KEY?

In F-PC, change the vectored function to the primitive, (KEY?) if you do not have a mouse.

Comment: The ideogram is another example of using deferred words to vector functions which can later be modified to meet application requirements.

L@

L! n addr seg --- NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Store integer n at offset addr in segment seg. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE L! ES POP BX POP ES: 0 [BX] POP NEXT

F-PC:

Not used.

Example:

HEX 1234 100 1780 L! DECIMAL

Store the hexadecimal value 1234 at offset 100 in segment 1780 and return to DECIMAL.

Comment: The F83 symbols in the ideogram for this function are reversed from those in MVP-FORTH and F-PC. Note: F83 also defines in a similar way LC! instead of !LC.

L@ addr seg --- n NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Fetch the integer value at offset addr in segment seg. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE L@ ES POP BX POP ES: 0 [BX] PUSH NEXT

F-PC:

Not used.

Example:

HEX 100 1780 L@ U. DECIMAL

Fetch the hexadecimal value at the long address offset 100 in segment 1780 and print it unsigned.

Comment: The F83 symbols in the ideogram for this function are reversed from those in MVP-FORTH and F-PC. Note: F83 also defines, in a similar way, LC@ instead of @LC.

LATEST --- addr MVP-FORTH

Leave the name field address of the top-most word in the CURRENT vocabulary. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

 : LATEST CURRENT @ @ ;

Fig: 8088/86:

 Same as MVP.

F83: 8088/86:

 Not used.

F-PC:

 Not used.

Example:

 LATEST ID.

This example causes the name of the top most ideogram in the CURRENT vocabulary to be printed.

Comment: Though not included in 79S it is usually available in most systems. Note that switching vocabularies for new definitions will change the address left by LATEST . Through the name field address at LATEST, one can find the code field address and write recursive routines.

LEAVE --- MVP-FORTH

Force termination of a DO-LOOP at the next LOOP or +LOOP by setting the loop limit equal to the current value of the index. The index itself remains unchanged, and execution proceeds normally until the loop terminating word is encountered. (*Fig*) Force termination of a DO-LOOP at the next opportunity by setting the loop limit equal to the current value of the index. The index itself remains unchanged, and execution proceeds normally until LOOP or +LOOP is encountered.

LEAVE

(79S) Same as MVP. (83S) Transfers execution to just beyond the next LOOP or +LOOP. The loop is terminated and loop control parameters are discarded. May only be used in the form: DO ... LEAVE ... LOOP or DO ... LEAVE ... +LOOP. LEAVE may appear within other control structures which are nested within the do-loop structure. More than one leave may appear within a do-loop. See; "9.3 Return Stack". (F83) I have to do this to be 83-Standard. (F-PC) Immediately exit a DO-LOOP.

Implementations:

MVP: 8088/86:

```
CODE LEAVE [BP] AX, MOV AX 2[BP], MOV NEXT
JMP END-CODE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE (LEAVE)
  LABEL PLEAVE 4 # RP ADD
  SS: 0 [RP] IP MOV RP INC RP INC NEXT
: LEAVE COMPILE (LEAVE) ; IMMEDIATE
```

F-PC:

```
CODE (LEAVE)
  MOV IP, 4 [BP] ADD RP, # 6 NEXT END-CODE
: LEAVE COMPILE (LEAVE) ; IMMEDIATE
```

Example:

```
: TEST 20000 0 DO I . ?TERMINAL
IF LEAVE THEN LOOP ;
```

This definition starts printing digits which can be stopped by typing any key.

Comment: Using LEAVE within a conditional structure within a DO-LOOP makes it function somewhat like a BEGIN-UNTIL with an escape clause.

The functions in 79S and 83S are different. One continues to execute the loop and leaves at the end where as the other jumps to the end. This difference would introduce a bug when porting from one system to the other if the necessary code changes are not made!

CAUTION — Executing LEAVE outside of a colon definition may crash your system.

LFA pfa --- lfa

MVP-FORTH

Convert the parameter field address of a dictionary definition to its link field address. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

 : LFA 4 - ;

Fig: 8088/86:

 Same as MVP.

F83: 8088/86:

 Not used.

F-PC:

 Not used.

Example:

 ' U* LFA U.

Find and print the link field address of U* , perhaps in anticipation of sealing the dictionary at that point.

Comment: Although this activity is forbidden by 79S, it is a convenient utility for moving around the header structure in a Forth development system. Its definition is dependent upon the implementation of the header structure.

LIMIT --- n

MVP-FORTH

A constant leaving the address just above the highest memory available for a disk buffer. Usually this is the highest system memory. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) The highest address in the Code Segment used by Forth.

Implementations:

MVP: 8088/86:

 EM CONSTANT LIMIT

Fig: 8088/86:

 Same as MVP.

F83: 8088/86:

 -2 CONSTANT LIMIT

F-PC:

LINK

Same as F83.

Example:

LIMIT U.

Examine the first location in memory not currently used by Forth.

Comment: Really, this is the limit for the Forth program in memory. In some implementations, this limit may not include all of RAM available. Forth can access all existing RAM addresses even if they are not within the confines of the Forth program. Such areas can be used for buffers by Forth. In MVP-FORTH, modifying the value in LIMIT and then executing CHANGE will change the upper bound of the Forth image in memory. In MVP-FORTH, its value may be altered and then CHANGE will dynamically reconfigure memory to the new value of LIMIT.

LINK

--- addr

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Points to the next task in the circular queue. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

VARIABLE LINK

F-PC:

Same as F83.

Example:

```
: TASK:  CREATE  TOS HERE #USER @ SMOVE
        HERE ENTRY LOCAL LINK !
        DUP HERE + DUP RPO ! 100 - SP0 ! SWAP UP !
        HERE #USER @ + HERE DP LOCAL !
        HERE SLEEP ALLOT ;
```

The example is a defining function by Henry Laxen.

Note: Not all of the words are in common usage. Refer to Henry Laxen's Multi-Tasking, Part I and Part II in Forth Dimensions Vol. V, Nos. 4, 5, 1983/1984

Comment: Depending upon the application, multi-tasking can be ac.

complied in a number of ways.

LINK>

NOT USED IN MVP-FORTH

addr1 --- addr2

(Fig) Not defined. (79S) Not defined. (83S *Definition Field Address Conversion Operators*) addr1 is the compilation address corresponding to the link field address addr1. (F83) Go from link field to code field. (F-PC) Go from link field address lfa to code field address cfa.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: LINK>      4 + H@    ;
```

F-PC:

```
: LINK>      L>NAME  NAME>  ;
```

Example:

```
'  DUP  DUP >LINK DUP LINK>  .S
```

A contrived example which goes from the code field address to the link field address and back, saving copies of the values and then displaying the values.

Comment: One of the several field address operators specified in a special word set of 83S.

LIST

n ---

MVP-FORTH

List the ASCII symbolic contents of screen n on the current output device, setting SCR to contain n. (Fig) Display the ASCII text of screen n on the selected output device. SCR contains the screen number during and after this process. (79S) Same as MVP. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

HEX

```
: LIST  CR  DUP  SCR  !
      ." SCR #" U.  10  0
```

LIT

```
DO CR I 3 .R SPACE I SCR @
.LINE ?TERMINAL
IF LEAVE THEN
LOOP CR ;
```

Fig: 8088/86:

HEX

```
: LIST    DECIMAL CR DUP SCR !    ." SCR # " .
10 0
DO CR I 3 .R SPACE I SCR @
.LINE ?TERMINAL IF LEAVE ENDIF LOOP CR ;
```

F83: 8088/86:

```
: LIST
1 ?ENOUGH CR DUP SCR ! .SCR L/SCR 0
DO CR I 3 .R SPACE DUP BLOCK I C/L
* + C/L -TRAILING >TYPE START/STOP
LOOP DROP CR ;
```

F-PC:

```
: LIST
ON> NEWBROWSE    ON> ?BROUSE    OFF> SEDING
OFF> NEWFL      1 ?ENOUGH =: LOADLINE <ED> ;
```

Example:

```
45 LIST
```

This causes Screen 45 to be printed.

Comment: This utility allows one to view the text contents in the selected screen.

LIT

--- n

MVP-FORTH

Within a colon-definition, LIT is automatically compiled before each 16-bit literal number encountered in input text. Later execution of LIT causes the contents of the next dictionary address to be pushed to the stack. (*Fig*) Same as MVP. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
CODE LIT [SI] AX, MOV SI INC SI INC APUSH
JMP END-CODE
```

Fig: 8088/86:

CODE LIT AX LODS APUSH JMP

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: LITERAL    STATE @
  IF COMPILE LIT , THEN ; IMMEDIATE
```

This example comes from the MVP-FORTH implementation.

Comment: An ideogram which tells the address interpreter that the next two bytes in the compiled definition are not an address to be executed but rather a numeric value to be pushed.

LITERAL

n ---

MVP-FORTH

If compiling, then compile the stack value n as a 16-bit literal, which when later executed, will leave n on the stack. (*Fig*) If compiling, then compile the stack value n as a 16-bit literal. This definition is immediate so that it will execute during a colon definition. The intended use is: : xxx [calculate] LITERAL ;. Compilation is suspended for the compile time calculation of a value. Compilation is resumed and LITERAL compiles the value. (79S) Same as MVP. (83S) Typically used in the form: [16b] LITERAL . Compiles a system dependent operation so that when later executed, 16b will be left on the stack. (F83) Compile the single integer from the stack as a literal. Now that code field of (LIT) is known, define LITERAL (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

```
: LITERAL    STATE @
  IF COMPILE LIT , THEN ; IMMEDIATE
```

Fig: 8088/86:

```
: LITERAL    STATE @
  IF COMPILE LIT , ENDIF ; IMMEDIATE
```

F83: 8088/86:

```
: LITERAL    COMPILE (LIT) , ; IMMEDIATE
```

F-PC:

```
: LITERAL    COMPILE (LIT) X, ; IMMEDIATE
```

LOAD

Example:

```
: ADD-RECORDLENGTH [ BLOCKSIZE
  RECORDS-PER-BLOCK / ] LITERAL + ;
```

A time-consuming calculation using constants is done once at compile time and placed into the definition by LITERAL. Note that BLOCKSIZE and RECORDS-PER-BLOCK used in the example must also be defined.

Comment: LITERAL, while harder to spell than LIT, is transportable, implementation independent and 79S.

LOAD

n ---

MVP-FORTH

Begin interpretation of screen n by making it the input stream; preserve the locators of the present input stream (from >IN and BLK). If interpretation is not terminated explicitly it will be terminated when the input stream is exhausted. Control then returns to the input stream containing LOAD, determined by the input stream locators >IN and BLK. (*Fig*) Begin interpretation of screen n. Loading will terminate at the end of the screen or at ;S. See ;S and -->. (79S) Same as MVP. (83S) The contents of >IN and BLK, which locate the current input stream, are saved. The input stream is then redirected to the beginning of screen u by setting >IN to zero and BLK to u. The screen is then interpreted. If interpretation from screen u is not terminated explicitly it will be terminated when the input stream is exhausted and then the contents of >IN and BLK will be restored. An error condition exists if u is zero. See: >IN BLK BLOCK (F83) Interpret a screen as if it were type[d] in . (F-PC) n1 is the line number of where to start loading.

Implementations:

MVP: 8088/86:

```
: LOAD 'LOAD @ EXECUTE ;
```

Fig: 8088/86:

```
:LOAD BLK @ >R IN @ >R
  0 IN ! B/SCR * BLK ! INTERPRET
  >R IN ! >R BLK ! ;
```

F83: 8088/86:

```
: (LOAD) FILE @ >R BLK @ >R >IN @ >R
  >IN OFF BLK ! IN-FILE @ FILE ! INTERPRET
  R> >IN ! R> BLK ! R> !FILES ;
: FLOAD
  IN-FILE @ >R (LOAD) R> FILE @ <>
  IF CR ." Returning to " FILE? SPACE
```

```

    THEN ;
    : NLOAD  DUP . FLOAD  ;
    DEFER LOAD
    ' NLOAD IS LOAD

```

F-PC:

```

    : LOAD
      DEPTH 1 < IF 1 THEN
        ?FILEOPEN  DUP>R  >LINE  CR  ." Loading.."
        <LOAD>  R>  =:  LOADLINE :

```

Example:

```

45  LOAD

```

This example will start loading the contents of Screen 45.

Comment: Screens which end with several blank lines will load faster if the ideogram, EXIT, appears following the last definition or operation. Also, one can avoid loading the whole screen without erasing the undesired contents by terminating the desired source with this ideogram. This technique is not sanctioned by 79S and is implementation dependent. This ideogram is vectored in MVP-FORTH for the benefit of the programmer. Normally, it invokes <LOAD>. Note that Screen zero is unloadable.

LOOP

MVP-FORTH

Increment the DO-LOOP index by one, terminating the loop if the new index is equal to or greater than the limit. The limit and index are signed numbers in the range -32,768..32,767. (*Fig*) Occurs in a colon-definition in form: DO ... LOOP. At run-time, LOOP selectively controls branching back to the corresponding DO based on the loop index and limit. The loop index is incremented by one and compared to the limit. The branch back to DO occurs until the index equals or exceeds the limit; at that time, the parameters are discarded and execution continues ahead. At compile-time, LOOP compiles (LOOP) and uses addr to calculate an offset to DO. n is used for error checking. (79S) Same as MVP. (83S) Increments the DO-LOOP index by one. If the new index was incremented across the boundary between limit-1 and limit the loop is terminated and loop control parameters are discarded. When the loop is not terminated, execution continues to just after the corresponding DO. sys is balanced with its corresponding DO. See: DO (F83) Not defined. (F-PC) Terminate a loop structure. Increment loop index by one and repeat <loop-body> until loop index crosses the boundary between limit and limit - 1. Used in the form DO ...

M*

LOOP.

Implementations:

MVP: 8088/86:

```
: LOOP 3 ?PAIRS COMPILE <LOOP>
  HERE - , ; IMMEDIATE
```

Fig: 8088/86:

```
: LOOP 3 ?PAIRS COMPILE (LOOP) BACK ;
```

F83: 8088/86:

```
: LOOP
  COMPILE (LOOP) 3000 ?PAIRS
  DUP 2+ <RESOLVE >RESOLVE ; IMMEDIATE
```

F-PC:

```
: LOOP
  COMPILE (LOOP) 2DUP 2+
  ?<RESOLVE ?>RESOLVE ; IMMEDIATE
```

Example:

```
: TEST 10 0 DO I . LOOP ;
```

This definition will cause the digits 0 through 9 to be printed.

Comment: This ideogram completes a DO-LOOP control structure. In the 79S and MVP-FORTH, the comparison uses signed arithmetic which may lead to problems if addresses are used as the parameters. The DO-LOOP structure can also be terminated with +LOOP or /LOOP. Both of these take an explicit increment from the stack; the latter uses unsigned arithmetic.

M*

n1 n2 --- d

MVP-FORTH

A mixed magnitude math operation which leaves the double number signed product to two signed numbers. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: M* DDUP XOR >R ABS SWAP ABS
  U* R> D+- ;
```

Fig: 8088/86:

```
: M* 2DUP XOR >R ABS SWAP ABS
  U* R> D+- ;
```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

45 2 M* D.

The two single precision values are placed on the stack and operated upon to leave a double precision value which is then printed.

Comment: With this ideogram, overflow is impossible. Use U* for full precision unsigned multiplication.

M* / d1 n1 n2 --- d2 MVP-FORTH

Multiplies d1 by n1 and divides the triple precision product by n2 leaving the quotient d2. All values are signed. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: M*/ DDUP XOR SWAP ABS >R SWAP
  ABS >R OVER XOR ROT ROT DABS
  SWAP R@ U* ROT R> U* ROT 0 D+
  R@ U/MOD ROT ROT R> U/MOD
  SWAP DROP SWAP ROT D+- ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: INCM 254 100 M*/ ;
```

This example converts inches in double precision to centimeters, also in double precision. The result is rounded toward zero.

Comment: This ideogram is most useful in maintaining precision in double number arithmetic. Its scaling capabilities frequently avert the expense of a full floating point math package. The notation in the book,

M/

Starting Forth, showing the divisor to be unsigned is in error.

M+ d1 n --- d2 MVP-FORTH

Add d1 to n and return d2. Note all values are signed. (*Fig*) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: M+    S->D   D+   ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
3456.   4   M+   D.
```

The value 3456 is entered as a double precision number by appending a decimal point, and then the single precision value of 4 is entered. The latter is converted to a double precision value and summed by the operator, leaving the double precision value on the stack to be printed.

Comment: This ideogram relieves some of the inconvenience of double precision arithmetic. Note that carry and overflow are ignored.

M/ d n1 --- n2 n3 MVP-FORTH

A mixed magnitude math operator which leaves the signed remainder n2 and signed quotient n3, from a double number dividend and divisor n1. The remainder takes its sign from the dividend. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: M/    OVER >R >R DUP D+- R@ ABS U/MOD
      R@ XOR +- SWAP R> +- SWAP ;
```

Fig: 8088/86:

```
: M/    OVER >R >R DABS R ABS U/    R>
```

R XOR +- SWAP R> +- SWAP ;

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

45. 7 M/ . .

Enter a double precision integer by ending the value with a decimal point, and then the single precision value followed by the operator. Then print the quotient followed by the remainder.

Comment: An integer operator which permits greater precision than simple integer division. Note that the remainder is left on the stack according to the definition though it is not implied in the ideogram. Overflow if it occurs is ignored, as is division by zero. Note that the implementation used in MVP-FORTH is taken from fig-Forth and differs from that given in *Starting Forth*.

M/MOD

ud1 u2 --- u3 ud4

MVP-FORTH

An unsigned mixed magnitude math operation which leaves a double quotient ud4 and remainder u3, from a double dividend ud1 and single precision divisor u2. (*Fig*) Same as MVP. (*79S*) Not defined. (*83S*) Not defined. (*F83*) divides a double by a single, leaving a single quotient and a single remainder. Division is floored. (*F-PC*) Same as F83. Source code indicated the dividend is signed!

Implementations:

MVP: 8088/86:

```
: M/MOD
  >R 0 R@ U/MOD R> SWAP >R
  U/MOD R> ;
```

Fig: 8088/86:

```
: M/MOD
  >R 0 R U/ R> SWAP >R U/ R> ;
```

F83: 8088/86:

```
: M/MOD
  ?DUP
  IF DUP >R 2DUP XOR >R >R
  DABS R@ ABS UM/MOD SWAP R> ?NEGATE
```

MAX

```
SWAP R> 0<
  IF NEGATE OVER
    IF 1- R@ ROT - SWAP
  THEN THEN
R> DROP
THEN ;
```

F-PC:

```
: M/MOD
  ?DUP
  IF dup>r 2DUP XOR >R >R DABS R@ ABS UM/MOD
    SWAP R> ?NEGATE
    SWAP R> 0<
    IF NEGATE OVER
      IF 1- R@ ROT - SWAP THEN
    THEN r>drop
  THEN ;
```

Example:

```
45. 7 M/MOD # #S # TYPE SPACE U.
```

Enter a double precision integer by ending the value with a decimal point and follow it by a single precision value. The quotient is then printed followed by the remainder. In the absence of a double precision unsigned output operator, numerical conversion is used in the example.

Comment: This fig-Forth function does not conform with the convention for the prefix “M” which implies the use of signed values. In this case, the input values are unsigned. Note that this ideogram is very similar to M/ — both leave the remainder and quotient, but M/ is signed. When porting code, the difference and a change to floored division will introduce obscured bugs.

MAX

n1 n2 --- n3

MVP-FORTH

Leave the greater of two numbers. (*Fig*) Same as MVP. (*79S*) Same as MVP. (*83S*) n3 is the greater of n1 and n2 according to the operation of >. (*F83*) Return the maximum of n1 and n2. (*F-PC*) Same as F83.

Pronounced: max

Implementations:

MVP: 8088/86:

```
: MAX DDUP <
  IF SWAP THEN DROP ;
```


Fig: 8088/86:

```
: MAX    2DUP <
  IF SWAP  ENDIF  DROP ;
```

F83: 8088/86:

```
CODE MAX
  AX POP    DX POP    DX AX CMP    0<
  IF  DX AX XCHG  THEN  1PUSH
```

F-PC:

```
CODE MAX
  POP AX    POP BX    CMP BX, AX  <=
  IF  1PUSH  THEN
  PUSH BX    NEXT    END-CODE
```

Example:

```
4 3 MAX .
```

Enter two single precision values on the stack, leave only the maximum value, and print it.

Comment: Provides a simple way of putting a lower limit on numbers. A signed comparison is used.

MAX-DRV

--- n

MVP-FORTH DISK I-O

A constant which returns the current maximum number of drives. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementation:

MVP: 8088/86:

```
2 CONSTANT MAX-DRV
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
MAX-DRV .
```

This example will print the maximum number of drives for which the system is presently configured.

MESSAGE

Comment: In some implementations of Forth this value is fixed and not available to the programmer. However, in other implementations the value may be changed from time to time. It therefore needs to be placed in the Forth vocabulary. In MVP-FORTH provisions are made for up to 5 drives. More than five drives cannot be accomodated without re-cross-compiling your system. It will require extending the array, DEN.

MESSAGE n --- NOT USED IN MVP-FORTH

(Fig) Print on the selected output device the text of line n relative to screen 4 of drive 0. n may be positive or negative. MESSAGE may be used to print incidental text such as report headers. If WARNING is zero, the message will simply be printed as a number (disk un-available). (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

(This ideogram is not implemented in MVP-FORTH.)

Fig: 8088/86:

```
: MESSAGE     WARNING @
  IF 2DUP
    IF 4 OFFSET @     B/SCR */ - .LINE     SPACE     ENDIF
  ELSE     ." MSG # " .
  ENDIF ;
```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

12 MESSAGE

Line 12 of Screen 4 would be printed on the output device.

Comment: This provides one method of accessing error and warning messages without having the text reside in memory. However, it is often more efficient and really takes little more space to place the required error messages in line as in MVP-FORTH. The convention in fig-Forth of having the text of messages on screen 4, often overlaps binary code used to boot up the system on drive 0 .

$$n_1 \quad n_2 \quad \text{---} \quad n_3$$

MVP-FORTH

Leave the lesser of two numbers. (*Fig*) Same as MVP. (*79S*) Same as MVP. (*83S*) $n3$ is the lesser of $n1$ and $n2$ according to the operation of $<$. (*F83*) Return the minimum of $n1$ and $n2$. (*F-PC*) Same as F83.

Pronounced: min

Implementations:

MVP: 8088/86:

```
: MIN    DDUP  >
  IF SWAP THEN DROP ;
```

Fig: 8088/86:

```
: MIN    2DUP  >
      IF SWAP ENDIF    DROP ;
```

F83: 8088/86:

```
CODE MIN
  AX POP    DX POP    AX DX CMP    0<
  IF    DX AX XCHG    THEN    1PUSH
```

F-PC:

```
CODE MIN
  POP AX    POP BX    CMP BX, AX    <=
  IF      PUSH BX      NEXT
  THEN    1PUSH      END-CODE
```

Example:

4 2 MIN .

Enter two single precision integers and leave only the minimum value on the stack which is then printed.

Comment: Provides a way of placing a ceiling on a number, such as the maximum number of drives. A signed comparison is used.

n1 --- n2

NOT USED IN MVP-FORTH

(Fig) Leave the two's complement of a number. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86: Could be defined as:

: MINUS NEGATE ;

Fig: 8088/86:

MOD

CODE MINUS AX POP AX NEG APUSH JMP

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

45 MINUS .

Enter an integer and then negate it and print the negative value.

Comment: This now obsolete fig-Forth ideogram has been replaced by NEGATE in 79S. Note that its pronunciation could be confused with the ideogram which does a subtract operation. It may appear in older programs.

MOD

n1 n2 --- n3

MVP-FORTH

Divide n1 by n2, leaving the remainder n3, with the same sign as n1. (*Fig*) Leave the remainder of n1/n2, with the same sign as n1. (79S) Same as MVP. (83S) n3 is the remainder after dividing n1 by the divisor n2. n3 has the same sign as n2 or is zero. An error condition results if the divisor is zero or if the quotient falls outside of the ranges {-32,768..32,767}. See: "division, floored". (F83) Not defined. (F-PC) Return the modulus of the numerator and denominator, where the quotient is "floored". This is designed to yield a result having the same sign as the denominator. Note that if the denominator is positive, the result is positive, regardless of the sign of the numerator.

Pronounced: mod

Implementations:

MVP: 8088/86:

: MOD /MOD DROP ;

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Same as MVP.

F-PC:

Same as MVP.

Example:

45 7 MOD .

Enter two integers, divide them leaving the remainder which is then printed.

Comment: A signed integer arithmetic operator. Note that division by zero is ignored; its result is unpredictable.

MON

NOT USED IN MVP-FORTH

(*Fig*) Exit to the system monitor, leaving a re-entry to Forth, if possible. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

(This ideogram has not been implemented in the MVP-FORTH. The ideogram BYE returns to the operating system rather than the monitor.)

Comment: Though defined in the *Installation Manual*, it is usually not implemented in fig-Forth. BYE will usually exit fig-Forth to the underlying operating system from which Forth's current image may be saved.

MOVE

addr1 addr2 n ---

MVP-FORTH

Move the specified quantity n of 16-bit memory cells beginning at addr1 into addr2. The contents of addr1 is moved first. If n is negative or zero, nothing is moved. (*Fig*) Move the contents of n memory cells (16-bit contents) beginning at addr1 into n cells beginning at addr2. The contents of addr1 is moved first. This definition is appropriate on word addressing computers. (79S) Same as MVP. (83S) Not defined. (F83) Move the specified bytes without overlapping. (F-PC) Move the specified bytes n1 from address a1 to address a2. No overlapping of data will occur.

Implementations:

MVP: 8088/86:

```
: MOVE
  0 MAX 2* <CMOVE> ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: MOVE
  -ROT 2DUP U<
```

MU/MOD

IF ROT CMOVE> ELSE ROT CMOVE THEN ;

F-PC:

Same as F83.

Example:

HERE PAD 20 MOVE

This example will move 40 bytes from the address of HERE to the address of PAD.

Comment: This ideogram moves two bytes at a time but otherwise is similar to CMOVE. Nothing is said about overlapping source and destination fields. It is probably best reserved for implementation on 16-bit machines where word boundaries may be important.

MU/MOD

NOT USED IN MVP-FORTH

d n - rem dquot

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Divides a double by a single, leaving a double quotient and a single remainder. Division is floored. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

: MU/MOD

>R 0 R@ UM/MOD R> SWAP >R UM/MOD R> ;

F-PC:

Same as F83.

Example:

1234567. 2 MU/MOD D. .

Enter the double number 24 and single number 6; execute the function; and print the double number quotient and single integer remainder.

Comment: Differs from UM/MOD which leaves only a single precision quotient. Note that both perform floored division.

MYSELF

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined. (*Common usage*) Compiles the cfa of the word currently being defined. Utilized to effect recursion.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Common usage:

: MYSELF LATEST PFA CFA , ; IMMEDIATE

Example:

```
: TEST    DUP . 1 - 0 OVER = NOT
  IF MYSELF ELSE EXIT THEN ;
10 TEST
```

Executing the TEST function after entering any integer such as 10, will type numbers from that integer to 1.

Comment: Recursive functions call themselves. Because the smudge bit is set while a function is being defined, the function will not be recognized until the definition is completed. Normally, direct recursion is not possible. An alternative function is RECURSE.

NAME>

NOT USED IN MVP-FORTH

addr1 --- addr2

(Fig) Not defined. (79S) Not defined. (83S *Definition Field Address Conversion Operators*) addr2 is the compilation address corresponding to the name field address addr1. (F83) Go from name field to code field. (F-PC) Go from name field address nfa to code field address cfa.

Implementations:

MVP: 8088/86:

Not used.

NEGATE

Fig: 8088/86:

Not used.

F83: 8088/86:

: NAME> 2- H@ ;

F-PC:

: NAME> 1 TRAVERSE 1+ Y@ ;

Example:

' DUP DUP >NAME DUP NAME> .S

A contrived example. Find the code field address of DUP, then go to the name field address and back, saving a copy of each along the way. Finally print the three values for comparison.

Comment: The ideogram is one of the 83S *Field Address Conversions Word Set*.

NEGATE

n --- -n

MVP-FORTH

Leave the two's complement of a number, i.e., the difference of 0 less n. (*Fig*) Not defined. (*79S*) Same as MVP. (*83S*) n2 is the two's complement of n1, i.e., the difference of zero less n1. (*F83*) Turn the number into its negative. A twos complement op. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

CODE NEGATE

AX POP AX NEG APUSH JMP END-CODE

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE NEGATE

AX POP AX NEG 1PUSH

F-PC:

Same as F83.

Example:

45 NEGATE .

Enter an integer and then convert to its negative value which is printed.

Comment: This 79S ideogram replaces MINUS in fig-Forth. Note that

HEX - 8000, when negated, has a value outside the range of signed single precision integers. No error condition is raised.

NEXT

MVP-FORTH

A constant pointing to the machine code entry point of the inner interpreter. (*Fig*) This is the inner interpreter that uses the interpretive pointer IP to execute compiled Forth definitions. It is not directly executed but is the return point for all code procedures. It acts by fetching the address pointed by IP, storing this value in register W. It then jumps to the address pointed to by the address pointed to by W. W points to the code field of a definition which contains the address of the code which executes for that definition. This usage of indirect threaded code is a major contributor to the power, portability, and extensibility of Forth. Locations of IP and W are computer specific. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) If the loop counter I is non-zero, decrement it and loop back the code following the matching FOR. If the loop counter is zero, discard it and continue execution.

Implementations:

MVP: 8088/86:

```
NEXT CONSTANT NEXT
```

(The cross-compiler uses the label NEXT to define the constant NEXT.)

Fig: 8088/86:

Not used.

F83: 8088/86:

Used as an ASSEMBLER macro.

F-PC:

```
CODE NEXT|
  SUB 0 [RP], # 1 WORD    U>=
  IF  MOV IP, ES: 0 [IP]  NEXT
  THEN
  ADD RP, # 2    ADD IP, # 2    NEXT    END-CODE
: COMPILE NEXT|    ?<RESOLVE    ;    IMMEDIATE
```

Example:

```
CODE DROP    H POP    NEXT JMP    END-CODE
```

This example is taken from the MVP-FORTH 8088/86 source code.

Comment: A memory location unique to the system implementation,

NFA

whose execution enters the inner interpreter. The machine code instruction NEXT JMP is the usual exit from a CODE definition.

NEXT1 ---

MVP-FORTH

A constant pointing to the entry point within the NEXT routine to be used by EXECUTE. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
  NEXT1  CONSTANT  NEXT1
  ( The cross-compiler uses the label NEXT1
    to define the constant NEXT1.)
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
CODE EXECUTE
  BX POP  NEXT1 JMP  END-CODE
```

This example comes from the MVP-FORTH 8080 source code.

Comment: A memory location unique to the system implementation, which causes the function of EXECUTE to begin.

NFA pfa --- nfa

MVP-FORTH

Convert the parameter field address of a definition to its name field. (*Fig*) Same as MVP. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: NFA  5  -  -1  TRAVERSE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
'  FORGET  NFA  ID.
```

In this contrived example, get the parameter field address of FORGET and then the name field address and print the name: FORGET.

Comment: This manipulation, though prohibited by 79S, is most convenient in development systems. It sets up the address for ID. .

NIP n1 n2 --- n2 NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Drop the second element from the stack. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
CODE NIP        AX POP        DX POP        1PUSH
```

F-PC:

```
CODE NIP
```

```
POP AX        ADD SP, # 2        1PUSH        END-CODE
```

Example:

```
1 2 NIP .S
```

Place the integer 2 on top of a 1 on the data stack. NIP will then remove the second on the stack as shown by .S.

Comment: This ideogram is commonly used. It amounts to SWAP DROP.

NOOP --- NOT USED IN MVP-FORTH

A Forth "no operation". (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) One of the most useful words in Forth. Does nothing. (F-PC) Same as F83.

NOT

Implementations:

MVP: 8088/86:

; NOOP ;

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

CODE NOOP NEXT

F-PC:

Same as F83.

Example:

NOOP

Entering this ideogram does nothing but return the prompt.

Comment: This is just an ideogram which does nothing except take up time and space which, in fact, is sometimes useful.

NOT

flag1 --- flag2

MVP-FORTH

Reverse the boolean value of flag1. This is identical to 0=. (*Fig*) Not defined. (79S) Same as MVP. (83S) 16b2 is the one's complement of 16b1. (F83) Does a ones complement of the top. Equivalent to -1 XOR. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

CODE NOT

AX POP AX AX, OR 1 AX, MOV NOT1 JZ AX DEC
HERE LABEL NOT1 APUSH JMP END-CODE

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE NOT

AX POP AX NOT 1PUSH

F-PC:

Same as F83.

Example:

0 NOT .

Enter a false flag 0 , make it true and print it.

Comment: NOT and 0= have the same function; however, depending on the context, one may be more readable than the other. Note: NOT is not the bitwise one's complement which has the *79S Reference Word Set* reference word set ideogram COM, (not implemented in MVP-FORTH). To confuse the user the ideogram was made a one's complement in the 83S.

NUMBER

addr --- d

MVP-FORTH

Convert the count and character string at addr, to a signed 32-bit integer, using the current base. If numeric conversion is not possible, an error condition exists. The string may contain a preceding negative sign. (*Fig*) Convert a character string left at addr with a preceding count, to a signed double number, using the current numeric base. If a decimal point is encountered in the text, its position will be given in DPL, but no other effect occurs. If numeric conversion is not possible, an error message will be given. (*79S Reference Word Set*) Same as MVP. (*83S*) Not defined. (*F83*) Convert a string to a number. Normally (NUMBER) (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

```
: NUMBER 'NUMBER @ EXECUTE ;
```

Fig: 8088/86:

HEX

```
: NUMBER 0 0 ROT DUP 1+ C@ 2D = DUP >R + -1 DPL
  BEGIN ! (NUMBER) DUP C@ BL -
  WHILE DUP C@ 2E - 0 ?ERROR 0
  REPEAT DROP R>
  IF D- ENDIF ;
```

F83: 8088/86:

DEFER NUMBER

```
: (NUMBER) NUMBER? NOT ?MISSING ;
' (NUMBER) IS NUMBER
```

F-PC:

DEFER NUMBER

```
: (NUMBER) %NUMBER NOT ?MISSING ;
' (NUMBER) IS NUMBER
```

Example:

```
: ?VALUE ." INPUT A VALUE --- "
  QUERY BL WORD NUMBER QUIT ;
```

NUMBER?

This example prompts for a value, then converts the value to a double precision number, placing it on the stack. It could be printed with D. .

Comment: The version and implementation of this ideogram in MVP-FORTH conforms with that in the *79S Reference Word Set* and *fig-Forth*. It will recognize two non-numeric characters: a decimal point and a leading negative sign. The position of the decimal point is recorded in the user variable DPL. This feature enables a user program to scale or adjust the converted value as desired. It will give an error message if any other special character is used. Note that the definition given in *Starting Forth* is different. Because of the possible variations, this ideogram utilizes an execution vector which will permit easy redefinition. It defaults to <NUMBER>.

NUMBER? addr --- d f NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Convert the count delimited string at addr to a double number. NUMBER? takes into account a leading minus sign, and stores a pointer to the last delimiter in DPL. The string must end with a blank. Leaves a true flag if successful. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

DEFER NUMBER?

' (HNUMBER?) IS NUMBER?

F-PC:

: NUMBER?

FALSE OVER COUNT BOUNDS

?DO I C@ BASE @ DIGIT NIP

IF DROP TRUE LEAVE THEN

LOOP

IF (NUMBER?) ELSE DROP 0 0 FALSE THEN ;

Example:

: TEST " 123456 " ;

' TEST 5 + NUMBER? . D.

TEST is one way to put a counted string at a known place in memory.

You could examine it with DUMP. Then execute the ideogram NUMBER? and output the flag and the value.

Comment: The ideogram may be used to convert a counted string to its double precision integer value.

OCTAL

MVP-FORTH SUPPLEMENTAL

Set the number conversion base to decimal 8. (*Fig*) Not defined. (*79S Reference Word Set*) Same as MVP. (*83S*) Not defined. (*F83*) All subsequent numeric I-O will be in Octal. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

```
: OCTAL 8 BASE ! ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Same as MVP.

F-PC:

Same as MVP.

Example:

```
HEX FF OCTAL .
```

Input the hex value FF and print its octal equivalent.

Comment: If octal values are to be used this is the ideogram to use to set the appropriate value of BASE .

OF

NOT USED IN MVP-FORTH

```

          --- addr n      (compiling)
n1 n2 --- n1             (if no match)
n1 n2 ---                (if there is a match)
```

(*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined (*F83*) Not defined. (*F-PC*) SeeCASE. (*Eaker*) Used in a colon definition in the form: CASE...OF...ENDOF...ENDCASE. Note that OF...ENDOF pairs may be repeated as necessary. At run-time, OF checks n1 and n2 for equality. If equal, n1 and n2 are both dropped from the stack, and execution continues to the next ENDOF. If not equal, only n2 is dropped, and execution jumps to whatever follows the next ENDOF. At compile-time, OF emplaces (OF) (an assembly code function if used) and

OFF

reserves space for an offset at addr. addr is used by END OF to resolve the offset. n is used for error checking.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

: OF

COMPILE (OF) >MARK ; IMMEDIATE

Eaker:

: OF

4 ?PAIRS COMPILE OVER COMPILE = COMPILE 0BRANCH
HERE 0 , COMPILE DROP 5 ; IMMEDIATE :

Example:

: GEN CASE

0 OF IMMEDIATE END OF

10 OF DIRECT END OF

20 OF INDEXED END OF

30 OF EXTENDED END OF

MODE-ERROR

ENDCASE RESET ;

The example is taken from the original article by Charles E. Eaker. The MODE-ERROR and RESET functions are taken from his implementation of fig-Forth and are not in common usage. They could be omitted from the example.

Comment: This is probably the most commonly used CASE function. It was published in *Forth Dimensions* Vol. II, No. 3, 1980 as one of three winners of FIG's CASE Statement Contest. All entries to the contest were published in that issue.

OFF

addr ---

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Set the contents of addr to FALSE. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE OFF BX POP FALSE # 0 [BX] MOV NEXT

F-PC:

CODE OFF
POP BX MOV 0 [BX], # FALSE WORD
NEXT END-CODE

Example:

```
: 0? 0 = PAD SWAP IF ON ELSE OFF THEN ;
```

Define a function to test the top of the stack for zero and set the value at PAD accordingly.

Comment: Rather than testing the top of the stack, one could test an I-O controller port to see if a device was on or off and set the value of a variable accordingly.

OFFSET

--- addr

MVP-FORTH

A variable that contains the offset added to the block number on the stack by BLOCK to determine the actual physical block number. The user must add any desired offset when utilizing BUFFER. (*Fig*) A user variable which may contain a block offset to disc drives. The contents of OFFSET is added to the stack number by BLOCK. Messages by MESSAGE are independent of OFFSET. See BLOCK, DR0, DR1, MESSAGE (*79S Reference Word Set*) Same as MVP. (*83S*) Not defined. (*F83*) Added to all block references. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

HEX 48 USER OFFSET

Fig: 8088/86:

HEX 1E USER OFFSET

F83: 8088/86:

16 USER VARIABLE OFFSET

F-PC:

ONLY

VARIABLE OFFSET

Example:

```
: DR0    0    OFFSET    !    ;
```

This definition sets the value of the user variable, `OFFSET`, to zero, such that subsequent Screen accesses will be to Drive 0.

Comment: This variable is available in a variety of implementations of Forth and is convenient not only for accessing different disk drives but for setting a base BLOCK from which data can be read for purposes of drive selection. However, using the ideograms DR0, DR1, etc. is safer, easier and more convenient.

ON addr --- NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Set the contents of addr to TRUE. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE ON BX POP TRUE # 0 [BX] MOV NEXT

F-PC:

CODE ON

```
POP BX      MOV 0 [BX], # TRUE WORD
```

NEXT END-CODE

```
: 0?    0 = PAD SWAP IF ON ELSE OFF THEN ;
```

Define a function to test the top of the stack for zero and set the value at PAD accordingly.

Comment: Rather than testing the top of the stack, one could test an I-O controller port to see if a device was on or off and set the value of a variable accordingly.

ONLY --- NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined (F83) Not defined. (F-PC) Clean off the vocabulary stack, leaving only ROOT. (Ragsdale) Select just the ONLY vocabulary as both the transient

vocabulary and resident vocabulary in the search order.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

: ONLY

```
[ ' ] ROOT >BODY CONTEXT #VOCS 1- 2*
2 DUP ERASE + ! ROOT ;
```

F-PC:

: ONLY

```
[ ' ] ROOT >BODY CONTEXT #VOCS 1- 2*
2DUP ERASE + ! ROOT ;
```

Example:

ONLY FORTH

Make FORTH the only operative vocabulary.

Comment: This ideogram is a part of an alternative to the fig-Forth and F79 vocabulary structures. It was introduced by William F. Ragsdale at the 1982 FORML Conference. A number of implementations have been adopted.

OR

n1 n2 --- n3

MVP-FORTH

Leave the bitwise-inclusive or of two numbers. (*Fig*) Leave the bitwise logical or of two 16-bit values. (*79S*) Same as MVP. (*83S*) 16b3 is the bit-by-bit inclusive-or of 16b1 with 16b2. (*F83*) Returns the bitwise OR of n1 and n2 on the stack. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

```
CODE OR   AX POP   BX POP   BX AX, OR   APUSH JMP
END-CODE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE OR   BX POP   AX POP   BX AX OR   1PUSH
```

F-PC:

ORDER

Same as F83.

Example:

```
HEX
20 30 OR
DECIMAL
```

The bit pattern associated with 20 and 30 will be logically OR'ed together and left on the stack: 30 .

Comment: An ideogram which performs a logical operation common in computing.

ORDER

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined (F83) Not defined. (F-PC) Displays the contents of the CONTEXT vocabulary stack. (Ragsdale) List the vocabulary names forming the search order in their present search order sequence. Then show the vocabulary into which new definitions will be placed.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: ORDER
  CR ." Search: " CONTEXT #VOCS 0
  DO DUP @ ?DUP
    IF BODY> >NAME .ID THEN 2+
  LOOP DROP
  CR ." Compile: " CURRENT @ BODY> >NAME .ID ;
```

F-PC:

```
: ORDER
  CR ." Context: " CONTEXT #VOCS 0
  DO DUP @ ?DUP
    IF BODY> >NAME .ID THEN 2+
  LOOP DROP
  CR ." Current: " CURRENT @ BODY> >NAME .ID ;
```

Example:

```
ORDER
```

The format of the display is implementation dependent.

Comment: This ideogram is a part of an alternative to the fig-Forth and F79 vocabulary structures. It was introduced by William F. Ragsdale at the 1982 FORML Conference. A number of implementations have been adopted.

OUT --- addr MVP-FORTH

A user variable that contains a value incremented by EMIT. The user may alter and examine OUT to control display formatting. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Assembler function only.

Implementations:

MVP: 8088/86:

HEX 4A USER OUT

Fig: 8088/86:

HEX 1A USER OUT

F83: 8088/86:

```
: OUT ( # apf --- )
  CR ." Subscript out of range on " DUP
  BODY> >NAME .ID ." Max is " ?
  ." tried " . QUIT ;
```

F-PC:

Not used.

Example:

OUT @ .

Examine the present value in the user variable, OUT, and print it.

Comment: This ideogram, though not present in all implementations, is useful for checking space remaining on a line of output. EMIT always increments it by one, even for nonprinting control characters.

OVER n1 n2 --- n1 n2 n1 MVP-FORTH

Leave a copy of the second number on the stack. (*Fig*) Copy the second stack value, placing it as the new top. (79S) Same as MVP. (83S) 16b3 is a copy of 16b1. (F83) Copy the second element to the top. (F-PC) Same as F83.

P!

Implementations:

MVP: 8088/86:

```
CODE OVER    DX POP    AX POP    AX PUSH    DPUSH JMP
END-CODE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE OVER    DX POP    AX POP    AX PUSH    2PUSH
```

F-PC:

```
CODE OVER
NOV DI, SP    PUSH 2 [DI]    NEXT    END-CODE
```

Example:

```
45 7 OVER
```

Enter two integers and then make a copy of the first one entered on the top of the stack leaving three values on the stack.

Comment: This is one of the most common stack operators.

P!

b n ---

MVP-FORTH

Output byte b to port n on an Intel based system. (*Fig - 8086/88 only*) Stores (outputs) a 16-bit number "n" at the I-O port. (Uses PC! for byte store.) (*79S*) Not defined. (*83S*) Not defined. (*F83*) Store a 16-bit word into an io port (*F-PC*) Write the value n1 to the 16-bit port#.

Pronounced: port store

Implementations:

MVP: 8088/86:

```
CODE P!
DX POP    AX POP    AL DX, OUT    NEXT JMP    END-CODE
```

Fig: 8088/86:

```
CODE P!
DX POP    AX POP    AX DX, OUT    NEXT JMP
```

F83: 8088/86:

```
CODE P!
DX POP    AX POP    DX AX OUT    NEXT
```

F-PC:

Same as F83.

Example:

```
HEX  41  E0  P!  DECIMAL
```

Go to hex and output the ASCII value of the character, A, on port E0 and return to decimal.

Comment: Gives access to any port of an Intel based system. Thus it is possible to write any value to any port. The use of this is obviously hardware dependent. In MVP-FORTH, the ideogram has maintained the 8-bit function to be compatible with other MVP implementations and PW! for word output..

NOTE: The output data may be either 8 bits or 16 bits depending on the implementation. If the AX register contains only a byte value, F83 and F-PC would put out the byte, however, specifically for byte values they use PC!

P@

n --- b

MVP-FORTH

Inputs byte b from port n on an Intel based system. (*Fig - 8086/88 only*) Fetches (inputs) a 16-bit value "n" from the I-O port. (Uses P@ for byte fetch.) (79S) Not defined. (83S) Not defined. (F83) Fetch a 16-bit word from an io port (F-PC) Read the 16-bit port# and return value n1.

Pronounced: port fetch

Implementations:

MVP: 8088/86:

```
CODE P@
  DX POP    DX AL, IN    AH AH, SUB
  APUSH JMP  END-CODE
```

Fig: 8088/86:

```
CODE P@
  DX POP    DX AX, IN    APUSH JMP
```

F83: 8088/86:

```
CODE P@
  DX POP    DX AX IN    1PUSH
```

F-PC:

```
CODE P@
  POP DX    IN AX, DX    PUSH AX
  NEXT      END-CODE
```

PC!

Example:

```
HEX E0 P@ DECIMAL .
```

Go to hex and get the value presently on port E0 and return to decimal and print that value.

Comment: Gives access to any port of an Intel based system. The port is read and its content placed on the top of the stack. In MVP-FORTH, the ideogram has maintained the 8-bit function to be compatible with other MVP implementations and PW@ for word input.

NOTE: The output data may be either 8 bits or 16 bits depending on the implementation. F83 and F-PC use the ideogram PC@ to distinguish the input of an 8-bit value.

PC! n port# --- NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Store an 8-bit byte into an input-putput port. (F-PC) Write the byte n1 to the 8-bit port#.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
CODE PC!        DX POP        AX POP        DX AL OUT        NEXT
```

F-PC:

```
CODE PC!  
POP DX        POP AX        OUT DX, AL  
NEXT        END-CODE
```

Example:

```
HEX 41 3F8 PC! DECIMAL
```

The example will output the ASCII value of A to the usual port address of COM1: on an IBM or clone system.

Comment: The ports on an IBM system may use 16 bits though only 8 bits are usually used.

PC@ port# --- n1 NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Fetch an 8-bit byte into an io port. (F-PC) Read the 8-bit port# and return the value n1.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE PC@

DX POP DX AL IN AH AH SUB 1PUSH

F-PC:

CODE PC@

POP DX IN AL, DX SUB AH, AH

PUSH AX NEXT END-CODE

Example:

HEX 3F8 PC@ DECIMAL

The example will fetch the 8-bit value from the usual port address of COM1: on an IBM or clone system.

Comment: The ports on an IBM system may use 16 bits though only 8 bits are usually used.

PAD --- addr MVP-FORTH

The address of a scratch area used to hold character strings for intermediate processing. The minimum capacity of PAD is 64 characters. (addr through addr+63) (Fig) Leave the address of the text output buffer, which is a fixed offset above HERE. (79S) Same as MVP. (83S) The lower address of a scratch area used to hold data for intermediate processing. The address or contents of PAD may change and the data lost if the address of the next available dictionary location is changed. The minimum capacity of PAD is 84 characters. (F83) Floating Temporary Storage area. (F-PC) - Floating Temporary Storage area.

Implementations:

MVP: 8088/86:

PAGE

```
HEX
: PAD
  HERE 44 + ;
```

Fig: 8088/86:
Same as MVP.

```
F83: 8088/86:
: PAD
  HERE 80 + ;
```

```
F-PC:
CODE PAD
  MOV BX, UP  MOV AX, DP [BX]  ADD AX, # 80
  1PUSH  END-CODE
```

Example:

```
HERE PAD 20 CMOVE
```

From the address of HERE to the address of PAD, move 20 bytes.

Comment: The location of PAD changes as the dictionary grows. Text strings build up from PAD while numbers are formatted downward.

PAGE

MVP-FORTH

Clear the terminal screen or perform an action suitable to the output device currently active. (*Fig*) Not defined. (*79S Reserved Word Set*) Same as MVP. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Printer dependent. Get to a new page. Increment the page number and reset the line number and the column number.

Implementations:

```
MVP: 8088/86:
: PAGE 'PAGE @ EXECUTE ;
```

Fig: 8088/86:
Not used.

```
F83: 8088/86:
: PAGE
  DOES> PERFORM 1 #PAGE +!
  #LINE OFF #OUT OFF ;
```

F-PC:
Same as F83.

Example:

PAGE

Entering this ideogram will clear the terminal screen by vectoring to <PAGE>.

Comment: There are many different terminals. Therefore, a new definition must be written and vectored into 'PAGE, before this ideogram will function properly. It should allow one to start with a clear screen and the cursor at home.

Note: The function is quite different in F83 and F-PC, where CLS has the same function.

PAUSE

MVP-FORTH UTILITY

Test the terminal keyboard for actuation of any key. If true, wait until a key has been pressed again. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Used by the Multitasker to switch tasks. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

```
: PAUSE ?TERMINAL
  IF 1000 0 DO LOOP
    BEGIN ?TERMINAL UNTIL KEY DROP
    2000 0 DO LOOP
  THEN ;
```

Fig: 8088/86:RNot used.

F83: 8088/86:

```
CODE PAUSE NEXT
```

F-PC:

```
CODE PAUSE \ Gets patched
  NOP NOP NOP NEXT END-CODE
```

Example:

```
: INDEX CR 1+ SWAP
DO CR I >R .R R> SPACES I .INDEX
  PAUSE ?TERMINAL IF LEAVE THEN 1
/LOOP ;
```

This example allows one to interrupt the INDEX listing by striking any key.

PC@

Comment: When PAUSE is followed by ?TERMINAL as in the example, striking any key once will continue the listing and striking any key twice in rapid succession will terminate the listing. The delay loops may have to be adjusted for individual systems and user preferences.

Note: F83, F-PC and some other Forth dialects use this ideogram with multitasking.

PC! b n --- MVP-FORTH

(Fig - 8088/86 only) Stores (outputs) an 8-bit number 'c' at the I-O port.. (79S) Not defined. (83S) Not defined. (F83) (F-PC) Write the byte n1 to the 8-bit port#.

Pronounced: port store

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

CODE PC!

DX POP AX POP AL DX, OUT NEXT JMP END-CODE

F83: 8088/86:

CODE PC!

DX POP AX POP DX AL OUT NEXT

F-PC:

Same as Fig.

Example:

HEX 41 E0 PC! DECIMAL

Go to hex and output the ASCII value of the character, A, on port E0 and return to decimal.

Comment: Gives access to any port of an Intel based system. Thus it is possible to write any value to any port. The use of this is obviously hardware dependent. The Fig - 8088/86 implementation uses PC! in place of P! used in MVP-FORTH and other implementations.

PC@ n --- b NOT USED IN MVP-FORTH

(Fig - 8086/88 only) Fetches (inputs) an 8-bit value "c" [n] from the I-O port. (79S) Not defined. (83S) Not defined. (F83) Not used. (F-PC) Not used.

Pronounced: port fetch

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

CODE PC@

DX POP DX AL, IN AH AH, SUB APUSH JMP

F83: 8088/86:

CODE PC@

DX POP DX AX IN 1PUSH

F-PC:

Same as Fig.

Example:

HEX E0 PC@ DECIMAL .

Go to HEX and get the value presently on port E0 and return to decimal and print that value.

Comment: Gives access to any port of an Intel based system. Any port on an 8088/86 CPU may be read and its content placed on the top of the stack. The Fig - 8088/86 implementation uses PC@ in place of P@ used in MVP-FORTH and other implementations.

PERFORM

cfa ---

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) the word whose code field is stored at the address pointed to by the number on the stack. Same as @ EXECUTE. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE PERFORM

W POP 0 [W] W MOV 0 [W] JMP END-CODE

F-PC:

CODE PERFORM

POP BX MOV AX, 0 [BX]

PICK

JMP AX END-CODE

Example:

```
VARIABLE TEMP
' NOOP TEMP !
TEMP PERFORM
```

The contrived example illustrates the way PERFORM can be used as a second level of indirection.

Comment: Some applications may make arrays of code field addresses which can be executed by indexing into them.

PFA nfa --- pfa MVP-FORTH

Convert the name field address of a compiled definition to its parameter field address. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: PFA    1   TRAVERSE   5 +   ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
HEX    4AC2   PFA    DECIMAL
```

Go to HEX and, presuming that the value, 4AC2, is a name field address, it will be replaced with its parameter field address.

Comment: Allows one to move around in the header of a definition. Although 79S forbids this, you sometimes have no alternative. See ID. .

PICK n1 --- n2 MVP-FORTH

Return the contents of the n1-th stack value, not counting n1 itself. An error condition results for n less than one. 2 PICK is equivalent to OVER. 1 .. n (*Fig*) Not defined. (79S) Same as MVP. (83S) 16b is a

copy of the +nth stack value, not counting +n itself. {0..the number of elements on stack-1. 0 PICK is equivalent to DUP; 1 PICK is equivalent to OVER. (F83) Reaches into the stack and grabs an element, copying it to the top of the stack. For example, if the stack has 1 2 3 Then 0 PICK is 3, 1 PICK is 2, and 2 PICK is 1. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

```
: PICK
  DUP 1 <
  ABORT" PICK ARGUMENT < 1" 2* SP@ + @ ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
CODE PICK
  BX POP    BX SHL    SP BX ADD
  SS: 0 [BX] AX MOV    1PUSH
```

F-PC:

```
CODE PICK
  POP DI    SHL DI, # 1    ADD DI, SP    PUSH 0 [DI]
  NEXT      END-CODE
```

Example:

```
2 PICK
```

This example is equivalent to OVER, although OVER is much faster.

Comment: Allows one to pick out of the stack any item and push a copy of it onto the top. In contrast with ROLL, PICK increases the stack depth by one, not counting the parameter eaten by PICK.

The function is one based in S79 and zero based in S83. Failure to make this distinction will cause trouble!

POP

NOT USED IN MVP-FORTH

(Fig) The code sequence to remove a stack value and return to NEXT . POP is not directly executable, but is a Forth re-entry point after machine code. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Assembler function only.

Implementations:

(This ideogram is not implemented in MVP-FORTH)

Comment: Though given in the *Installation Manual*, it is not available

PREV

in Forth implemented on an 8088/8086 or 8080/Z80 CPU.

PP n --- <text>

MVP-FORTH

On the latest screen listed, put <text> on line n. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```

: PP DUP FFFO AND
  ABORT" OFF SCREEN "
  1 TEXT PAD 1+ SWAP
  SCR @ <LINE> CMOVE UPDATE ;

```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```

99 LIST 99 CLEAR 99 LIST
0 PP ( THIS IS A NEW SCREEN )
99 LIST

```

After listing the screen, it is cleared with CLEAR and "(THIS IS A NEW SCREEN)" is put in line 0.

Comment: This implementation is adapted from the EDITOR included in the *fig-Forth Installation Manual*. Before loading an EDITOR, this ideogram marks it possible to enter new source text on a screen for loading. Without such an ideogram it is difficult to get started unless you are fortunate enough to have a disk already containing the source screens for an EDITOR. Note that this function is different from that described in *Starting Forth*.

```
PREV      --- addr
```

MVP-FORTH

A variable containing the address of the disk buffer most recently referenced. The UPDATE command marks this buffer to be later written to disk. (Fig) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Assembler function only.

*Implementations:***MVP: 8088/86:**

VARIABLE PREV FIRST PREV !

Fig: 8088/86:

BUF1 VARIABLE PREV

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

PREV @ U.

Get the value of this variable and print it.

Comment: The buffer management routines use PREV to monitor the most recently referenced buffer. It is seldom useful in applications.**PUSH**

NOT USED IN MVP-FORTH

(Fig) This code sequence stores machine registers to the computation stack and returns to NEXT. It is not directly executable, but is a Forth re-entry point after machine code. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.*Implementations:*

(This ideogram is not implemented in MVP-FORTH.)

Comment: This ideogram is not usually a part of an 8088/8086 or 8080/Z80 implementation of Forth. It is usually a part of the ASSEMBLER vocabulary. In MVP-FORTH, the equivalent entry point is named HPUSH, which should not be confused with DPUSH.**PUT**

NOT USED IN MVP-FORTH

(Fig) This code sequence stores machine register contents over the topmost computation stack value and returns to NEXT. It is not directly executable, but is a Forth reentry point after machine code. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Save the file currently in memory. If there is not enough disk space to save the current file, then prompts the user to insert another disk to save on. No return to editing is allowed, only ESC to DOS without saving is allowed if no new disk is inserted.

PW!

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

: PUT WRITE.FILE ;

Example:

PUT

Save the current file in memory.

Comment: The ideogram is present only in the original 6502 version of fig-Forth and in a completely different function in F-PC.

PW!

n1 n2 ---

MVP-FORTH

Output word n1 to port n2 on an 8086/8088 system. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

CODE PW!

DX POP AX POP AX DX, OUT NEXT JMP END-CODE

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

4567 PORT PW!

Output the 16-bit data word to a previously defined PORT.

Comment: Systems with 16-bit data busses may put out all 16 bits to one port. Since the ideogram P! as used in many 8-bit systems has the connotation of 8 bits already established, a new ideogram PW! has been

added in MVP-FORTH. Other implementations of Forth on 16-bit systems have changed the functional definition of P! to output 16 bits and added PC! to output 8 bits. The character connotation of 'C' seems wrong because a byte value is actually output. Furthermore, it seems better not to change established functional definitions.

PW@ n1 --- n2 MVP-FORTH

Input word n2 from port n1 on an 8086/8088 system. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

CODE PW@

DX POP DX AX,IN APUSH JMP END-CODE

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

PORT PW@ .

Input the 16 data bits on the previously defined PORT and print the value.

Comment: Systems with 16-bit data busses may input all 16 bits to one port. Since the ideogram P@ as used in many 8-bit systems has the connotation of 8 bits already established, a new ideogram PW@ has been added in MVP-FORTH. Other implementations of Forth on 16-bit systems have changed the functional definition of P@ to input 16 bits and added PC@ to input 8 bits. The character connotation of 'C' seems wrong because a byte value is actually input. Furthermore, it seems better not to change established functional definitions.

QUERY --- MVP-FORTH

Accept input of up to 80 characters (or until a 'return') from the operator's terminal, into the terminal input buffer. WORD may be used to accept text from this buffer as the input stream, by setting >IN and

QUIT

BLK to zero. (*Fig*) Input 80 characters of text (or until a "return") from the operators terminal. Text is positioned at the address contained in TIB with IN set to zero. (79S) Same as MVP. (83S) Not defined. (*F83*) Get more input from the user and place it at TIB. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

HEX

: QUERY TIB @ 50 EXPECT 0 >IN ! ;

Fig: 8088/86:

HEX

: QUERY TIB @ 50 EXPECT 0 IN ! ;

F83: 8088/86:

: QUERY

SPAN @ >R TIB 80 EXPECT SPAN @ #TIB !

BLK OFF >IN OFF R> SPAN ! ;

F-PC:

: QUERY

TIB COLS EXPECT SPAN @ #TIB ! >IN OFF ;

Example:

: TEST

." Input value --- " QUERY BL WORD NUMBER ;

This definition will print a prompt, then wait for an input value and place it on the stack.

Comment: Places a line of input in the terminal input buffer, where it may be processed by WORD , NUMBER , INTERPRET , or any user-defined routine.

QUIT

211 MVP-FORTH

Clear the return stack, setting execution mode, and return control to the terminal. No message is given. (*Fig*) Clear the return stack, stop compilation, and return control to the operators terminal. No message is given. (79S) Same as MVP. (83S) Clears the return stack, sets interpret state, accepts new input from the current input device, and begins text interpretation. No message is displayed. (*F83*) Defer the main loop. (*F-PC*) The main loop in Forth. Gets more input from the terminal and Interprets it. Responds with OK if healthy.

Implementations:
MVP: 8088/86:

```

: QUIT 0 BLK ! [COMPILE] [
  BEGIN CR RP! QUERY INTERPRET
    STATE @ NOT
    IF ." OK" THEN
  AGAIN ;

```

Fig: 8088/86:

```

HEX
: QUIT
  0 BLK ! [COMPILE] [
  BEGIN RP! CR QUERY INTERPRET STAT @ 0=
    IF ." OK" ENDIF
  AGAIN ;

```

F83: 8088/86:

```

: (QUIT)
  SP0 @ 'TIB ! BLK OFF [COMPILE] [
  BEGIN RP0 @ RP! STATUS QUERY INTERPRET
    STATE @ NOT
    IF ." ok" THEN
  AGAIN ;
  DEFER QUIT
  ' (QUIT) IS QUIT

```

F-PC:

```

: QUIT
  SP0 @ 'TIB ! [COMPILE] [
  BEGIN BEGIN RP0 @ RP! STATUS QUERY RUN
    STATE @ NOT
  UNTIL ." ok" AGAIN ;

```

Example:

```

." HELLO" QUIT

```

Print the text, HELLO, and then return to Forth without the usual prompt, OK.

Comment: Makes it possible to end the execution of a definition without having the ever-present OK produced. QUIT is also useful for user defined error message routines and application specific prompts.

R#

R --- n

NOT USED IN MVP-FORTH

(Fig) Copy the top of the return stack to the computation stack. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

: R I ;

F83: 8088/86:

Not used.

F-PC:

Not used.

Comment: This ideogram is now obsolete having been replaced by R@ in 79S.

R# --- addr

MVP-FORTH

A user variable which may contain the location of an editing cursor, or other file related function. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (*F83*) The cursor position during editing. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

HEX	4C	USER	R#
-----	----	------	----

Fig: 8088/86:

HEX 2E USER R#

F83: 8088/86:

VARIABLE R#

F-PC:

Same as F83.

Example:

R# @ .

Get the address of this user variable, fetch its value and print it.

Comment: This ideogram is generally only used when in the EDITOR

vocabulary. If it is not in the range of 0 .. 1023 , an error message of out of bounds is sometimes given.

```
R/W      addr blk f ---      MVP-FORTH
```

The fig-Forth standard disk read-write linkage. *addr* specifies the source or destination block buffer (not a 79S block buffer), *blk* is the sequential number of the referenced block; and *f* is a flag for *f* = 0 write and *f* = 1 read. *R/W* determines the location on mass storage, performs the read or write and performs any error checking. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: R/W  'R/W  @  EXECUTE  ;
```

Fig: 8088/86:

HEX

```

: R/W    USE @ >R SWAP SPBLK *    ROT USE !    SPBLK 0
DO OVER OVER T&SCALC SET-IO
  IF SEC-READ ELSE SEC-WRITE
  THEN 1+ 80 USE +!
LOOP DROP DROP R> USE ! ;

```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```

: BUFFER    USE @ DUP R
BEGIN +BUF UNTIL
USE ! R@ @ 0<R
IF R@ 2+ R@ @ 7FFF AND 0 R/W THEN
R@ ! R@ PREV ! R> 2+ ;

```

This definition is one of the principal uses of the ideogram, R/W.

Comment: This ideogram is a primitive in many implementations of Forth. If it is available, it is possible to read and write from disk to any area in memory such as a special buffer, without going through the regular block buffers. Of course such a procedure is installation dependent and prohibited by 79S. It is vectored for the benefit of the programmer, defaulting to the routine <R/W>.

F83: 8088/86:

```
CODE R>
  SS: 0 [RP] AX MOV   RP INC   RP INC   1PUSH
```

F-PC:

```
CODE R>
  PUSH 0 [RP]
  ADD RP, # 2
  NEXT   END-CODE
```

Example:

```
: TRY   >R   .   R>   ;
```

This example accomplishes the same thing as “SWAP”. Though it illustrates the ideogram, it would be poor programming practice.

Comment: An ideogram allowing manipulations of the return stack. Occasionally values from the data stack can be temporarily stored in the return stack and returned with this ideogram. However, this is a potentially dangerous procedure. Be careful to leave the return stack as you found it before exiting your definition, otherwise you will certainly crash the system.

R@

--- n

MVP-FORTH

Copy the number on the top of the return stack to the data stack. (*Fig*) Not defined. (79S) Same as MVP. (83S) 16b is a copy of the top of the return stack. (F83) Copies the value on the return stack to the parameter stack. (F-PC) Same as F83.

Pronounced: r-fetch

Implementations:

MVP: 8088/86:

```
CODE R@
  [BP] AX, MOV   APUSH JMP   END-CODE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
CODE R@
  SS: 0 [RP] AX MOV   1PUSH
```

F-PC:

```
CODE R@
  PUSH 0 [RP] NEXT   END-CODE
```

RECURSE

Example:

R@ .

Get the value present value on top of the return stack and print it.

Comment: This ideogram fetches the value on top of the return stack without changing it, and places it on the data stack. It is useful when using the return stack for temporary storage but be careful to finally leave the return stack as you found it. Otherwise, you will crash the system.

RECURSE

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Makes the definition this word is used in call itself at the point where it is used. ie, "RECURSION".

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

: RECURSE LAST @ NAME> , ; IMMEDIATE

F-PC:

: RECURSE

LAST @ NAME> X, ; IMMEDIATE

Example:

: TEST

DUP . 1 - 0 OVER = NOT

IF RECURSE ELSE EXIT THEN ;

10 TEST

Executing the TEST function after entering any integer such as 10, will type numbers from that integer to 1.

Comment: Recursive functions call themselves. Because the smudge bit is set while a function is being defined, the function will not be recognized until the definition is completed. Normally, direct recursion is not possible. An alternative function is MYSELF.

RECURSIVE

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Allow the current definition to be self referencing. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

: RECURSIVE REVEAL ; IMMEDIATE

F-PC:

Same as F83.

Example:

```
: TEST
  DUP . 1- 0 OVER = NOT
  IF RECURSIVE TEST ELSE EXIT THEN ;
10 TEST
```

The ideogram will then print the values from 10 zero.

Comment: This is an alternative to RECURSE, MYSELF or a variety of other methods of executing recursion.

REPEAT

MVP-FORTH

Used in a colon-definition. At run-time, REPEAT returns to just after the corresponding BEGIN. (Fig) Used within a colon-definition in the form: ... At run-time, REPEAT forces an unconditional branch back to just after the corresponding BEGIN. At compile-time, REPEAT compiles BRANCH and the offset from HERE to addr. n is used for error testing. (79S) Same as MVP. (83S) Used in the form: ... At execution time, REPEAT continues execution to just after the corresponding BEGIN. sys is balanced with its corresponding WHILE. See: BEGIN (F83) Not defined. (F-PC) Unconditional backward branch to just after BEGIN in a BEGIN <loop> flag WHILE <true> REPEAT loop.

Form: BEGIN ... WHILE ... REPEAT

Implementations:

MVP: 8088/86:

: REPEAT

REVEAL

```
>R >R [COMPILE] AGAIN R> R>  
2- [COMPILE] THEN ; IMMEDIATE
```

Fig: 8088/86:

```
: REPEAT  
>R >R [COMPILE] AGAIN R> R> 2 - ENDIF ;  
IMMEDIATE
```

F83: 8088/86:

```
: REPEAT  
2SWAP [COMPILE] AGAIN 2000 ?PAIRS <RESOLVE ;  
IMMEDIATE
```

F-PC:

```
: REPEAT COMPILE DOREPEAT ?<RESOLVE  
?>RESOLVE ; IMMEDIATE
```

Example:

```
: TEST 0  
BEGIN DUP . 9 <R  
WHILE 1+  
REPEAT ;
```

This definition illustrates another way of printing the digits 0 through 9.

Comment: REPEAT is a closing delimiter for a BEGIN control structure. Since it compiles an unconditional branch, the structure may be exited only at the corresponding WHILE.

REVEAL

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Replaces the Last definition in the Header Dictionary. (F-PC) Activates the Last definition in the Header Dictionary.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: REVEAL  
LAST @ DUP N>LINK SWAP 1+ H@ CURRENT @ HASH ! ;
```

F-PC:

```
: REVEAL
  LAST @ DUP N>LINK    SWAP CURRENT @ YHASH ! ;
```

Example:

```
: TEST
  DUP . 1- 0 OVER = NOT
  IF [ REVEAL ] TEST ELSE EXIT THEN :
10 TEST
```

Executing the TEST function after entering any integer such as 10, will type numbers from that integer to 1. Obviously a contrived example.

Comment: The ideogram is provides another way of executing recursive functions. Since it is not an IMMEDIATE function, it must be executed at compile time with the left and right brackets. It is made immediate with RECURSIVE, and has the same function as MYSELF and RECURSE.

ROLL

n ---

MVP-FORTH

Extract the n-th stack value to the top of the stack, not counting n itself, moving the remaining values into the vacated position. An error condition results for n less than one. 1 .. n (*Fig*) Not defined. (79S) Same as MVP. (83S) The +nth stack value, not counting +n itself is first removed and then transferred to the top of the stack, moving the remaining values into the vacated position. {0..the number of elements on the stack-1} 2 ROLL is equivalent to ROT; 0 ROLL is a null operation. (F83) Similar to SHAKE and RATTLE. Should be avoided. 1 ROLL is SWAP, 2 ROLL is ROT, etc. ROLL can be useful, but it is slow. (F-PC) Same as F83.

Form: 3 ROLL = ROT 1 ROLL = null operation

Implementations:

MVP: 8088/86:

```
: ROLL  DUP 1 <  ABORT" ROLL ARGUMENT 1"
  1+ DUP PICK SWAP 2*  SP@ +
  BEGIN DUP 2- @  OVER ! 2-  SP@
    OVER U< NOT
  UNTIL DDROP ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE ROLL

ROT

```
SS AX MOV    AX DS MOV    AX ES MOV
SI BX MOV    DI POP      DI SHL
DI CX MOV    SP DI ADD
DI SI MOV    SI DEC      SI DEC
0 [DI] AX MOV    AX PUSH
STD  REP MOVS  CLD  SP INC    SP INC
BX SI MOV    CS PUSH    DS POP    NEXT
```

F-PC:

```
: ROLL
>R R@ PICK    SP@ DUP 2+    R> 1+ 2* CMOVE>    DROP ;
```

Example:

```
2 ROLL
```

This example is the equivalent of the ideogram, SWAP.

Comment: This ideogram is the companion of PICK. While PICK copies its victim onto the top of the stack, ROLL moves it. PICK increases the stack depth by one while ROLL leaves it unchanged — not counting the parameter eaten by ROLL.

This function is one based in *F79* and zero based in *S83*. Failure to note this distinction will cause trouble!

ROT n1 n2 n3 --- n2 n3 n1 MVP-FORTH

Rotate the top three values, bringing the deepest to the top. (*Fig*) Rotate the top three values on the stack, bringing the third to the top. (*79S*) Same as MVP. (*83S*) The top three stack entries are rotated, bringing the deepest to the top. (*F83*) Rotate the top three element, bringing the third to the top. (*F-PC*) Same as *F83*.

Pronounced: rote

Implementations:

MVP: 8088/86:

```
CODE ROT
  DX POP    BX POP    AX POP
  BX PUSH    DPUSH JMP    END-CODE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE ROT
  DX POP    BX POP    AX POP    BX PUSH    2PUSH
```

F-PC:

Same as F83.

Example:

45 73 89 ROT

Enter three values and then take the first one in, 45, and move it to the top of the stack.

Comment: This ideogram performs the function of 3 ROLL, but rather more quickly.

RP!

MVP-FORTH

A computer dependent procedure to initialize the return stack pointer from user variable R0. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) (Warning, this is different from FIG Forth) Sets the return stack pointer to the specified value. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

CODE RP!

UP BX, MOV [BX] 8 + BP, MOV NEXT JMP END-CODE

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

CODE RP!

RP POP NEXT

F-PC:

Same as F83.

Example:

```
: QUIT 0 BLK ! [COMPILE] [  
  BEGIN CR RP! QUERY INTERPRET STATE @ NOT  
  IF ." OK" THEN  
  AGAIN ;
```

This example is taken from the MVP-FORTH implementation.

Comment: This ideogram is not a part of 79S, but may remain in the primitives.

RPP

RP@ --- addr MVP-FORTH

Leaves the current value in the return stack pointer register. (*Fig 8088/86*) Same as MVP. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Return the address of the next entry on the return stack. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

```
CODE RP@
  BP AX, MOV  APUSH JMP  END-CODE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE RP@
  RP AX MOV  1PUSH
```

F-PC:

```
CODE RP@
  PUSH RP  NEXT  END-CODE
```

Example:

```
RP@ @ U.
```

Nondestructively fetch and print the top item on the return stack.

Comment: Provides more general access to the insides of Forth, which is, of course, prohibited by 79S.

RPP --- addr MVP-FORTH

A constant returning a pointer to the cell in low memory which holds the Forth return stack pointer. It is used in the 8088/8086 or 8080/Z80 implementation. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
RPP CONSTANT RPP
( The cross-compiler uses the label RPP to define
  the constant RPP. )
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

RPP @ @ U.

This example achieves the same result as R@.

Comment: The cell at address RPP is a pseudo register for the simulated Forth machine. Since the implementation needs its internal registers for more time-critical activities, the return stack pointer resides at this fixed location in RAM.

S->D

n --- d

MVP-FORTH

Sign extend a single number to form a double number. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

CODE S->D

DX POP AX AX, SUB DX DX OR

STOD1 JNS AX DEC

HERE LABEL STOD1 DPUSH JMP END-CODE

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

22 S->D

Enter a single precision number on the stack and sign extend it to a double precision value. It is equivalent to entering twenty-two with a terminating decimal point (22.).

Comment: A convenient operation with integer arithmetic which maintains the correct value of the sign in the extension.

S>D

S0 --- addr MVP-FORTH

Returns the address of the bottom of the stack, when empty. (*Fig*) A user variable that contains the initial value for the stack pointer. See SP1 (79S Reference Word Set) Same as MVP. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Pronounced: s-zero

Implementations:

MVP: 8088/86:

: S0 SP0 @ ;

Fig: 8088/86:

6 USER S0

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

S0 U.

Get the address of the bottom of the stack and print it unsigned.

Comment: Allows one to reference the empty position of the data stack. It can be used to implement a nondestructive stack display such as .S. In fig-Forth, S0 is defined as a user variable to which MVP-FORTH has assigned the ideogram SP0, in order to free the use of S0 in accordance with 79S(R). *Starting Forth* appears to use the fig-Forth definition.

S>D n --- d NOT USED IN MVP-FORTH

(Fig) Not defined. **(79S)** Not defined. **(83S)** Not defined. **(F83)** Take a single precision number and make it double precision by extending the sign bit to the upper half of SCAN. **(F-PC)** Take a single precision number and make it double precision by extending the sign bit to the upper half.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE S>D AX POP CWD AX DX XCHG 2PUSH

F-PC:

Same as F83.

Example:

22 S>D D.

Enter a single precision number on the stack and sign extend it to a double precision value. It is equivalent to entering twenty-two with a terminating decimal point (22.).

Comment: A convenient operation with integer arithmetic which maintains the correct value of the sign in the extension. The ideogram makes a slight change in symbols from fig-Forth and MVP-FORTH in which S->D is used.

SAVE-BUFFERS

MVP-FORTH

Write all blocks to mass-storage that have been flagged as UPDATED. An error condition results if mass-storage writing is not completed. (Fig) Not defined. (79S) Same as MVP. (83S) The contents of all block buffers marked as UPDATEed are written to their corresponding mass storage blocks. All buffers are marked as no longer being modified, but may remain assigned. (F83) Write back all of the updated buffers to disk, and mark them as unmodified. Use this whenever you are worried about crashing or losing data. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: SAVE-BUFFERS
  #BUFF 1+ 0 DO 7FFF
  BUFFER DROP LOOP ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: SAVE-BUFFERS
  1 BUFFER# #BUFFERS 0
  DO DUP @ 1+
  IF DUP 6 + @ 0<
  IF DUP WRITE-BLOCK DUP 6 + OFF THEN 8 +
```

SAVE-FORTH

```
    THEN
  LOOP  DROP  ;
```

F-PC:

Not used.

Example:

SAVE-BUFFERS

Write all buffers marked by UPDATE back to the disks.

Comment: This ideogram replaces the older one FLUSH. Often the older ideogram is continued as at least an alias because it is very deeply embedded in Forth.

SAVE-FORTH

MVP-FORTH UTILITY

Save the current image of Forth on the default drive. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

DECIMAL

```
: SAVE-FORTH  FREEZE
  13 0 SYSCALL DROP  ( RESET DISK SYSTEM)
  14 0 SYSCALL DROP  ( SELECT DISK )
  CR CR ." FILE NAME ? ----"
  PAD 33 0 FILL PAD 1+ 11 BLANK
  QUERY  ( MAKE PAD FCB )
    46 WORD COUNT 8 MIN PAD 1+ SWAP CMOVE
    BL WORD COUNT 3 MIN PAD 9 + SWAP CMOVE
  19 PAD SYSCALL DROP  ( DELETE FILE )
  22 PAD SYSCALL DROP  ( MAKE FILE )
  256 BEGIN
    26 SWAP SYSCALL DROP  ( SET DMA ADDRESS)
    21 PAD SYSCALL DROP  ( WRITE SEQUENTIAL)
  128 + DUP HERE SWAP U< UNTIL DROP
  16 PAD SYSCALL DROP ;  ( CLOSE FILE )
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

CAUTION: You could destroy your present COM file.

SAVE-FORTH

Execution of this ideogram will prompt for the desired system's file name and erase it if it is present. It will then write the Forth image in memory to the file you selected. Drive 0 is selected.

Comment: This ideogram makes it possible to save a binary image of Forth at any time. It makes a rapid restart after a system crash possible. In this implementation all error messages from the system's BIOS are discarded. You may wish to modify the code to better suit your needs. It is particularly useful in turnkey systems which submit your Forth binary image upon booting up.

SCAN

NOT USED IN MVP-FORTH

addr len char --- addr' len'

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Given the address and length of a string, and a character to look for, run through the string until we find the character. Leave the address of the match and the length of the remaining string. (F-PC) Given the address and length of a string, and a character to look for, run through the string until we find the character. Leave the address of the match and the length of the remaining string.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE SCAN

```

AX POP    CX POP    $DONE JCXZ    DI POP
DS DX MOV  DX ES MOV  CX BX MOV
REP BYTE SCAS      0=
IF  CX INC  DI DEC  THEN
DI PUSH  CX PUSH  NEXT
```

F-PC:

SCR

```
CODE  SCAN
  POP AX      POP CX      JCXZ 0 $
  POP DI      MOV DX, ES   MOV ES, SSEG
  REPNZ  SCASB  MOV ES, DX   0-
  IF  INC CX  DEC DI  THEN
  PUSH DI      PUSH CX      NEXT
0 $:  PUSH CX      NEXT      END-CODE
```

Example:

```
PAD 20 EXPECT      ABCDEF GHIJK
PAD 20 ASCII G SCAN
```

The example places a number of characters at PAD. The scanning from PAD to the ASCII character returns the address and remaining characters on the stack.

Comment: The principle use of the ideogram is to scan the input stream as with PARSE in F-PC.

SCR

--- addr

MVP-FORTH

Leave the address of a variable containing the number of the screen most recently listed. The value of the variable is unsigned. (*Fig*) A user variable containing the screen number most recently reference by LIST. (*79S*) Same as MVP. (*83S*) Not defined. (*F83*) Holds the screen number last listed or edited. (*F-PC*) Not defined.

Pronounced: s-c-r

Implementations:

MVP: 8088/86:

HEX

4E USER SCR

Fig: 8088/86:

HEX

1C USER SCR

F83: 8088/86:

VARIABLE SCR

F-PC:

Not used.

Example:

```
: L  SCR  @  LIST  ;
```

This definition re-lists the last screen listed.

Comment: Using ! or +! to alter the contents of SCR is permitted. Some editors do this in commands which step forward or backward in a series of screens.

SEAL

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined (F83) Not defined. (F-PC) Remove all vocabularies from the CONTEXT vocabulary stack leaving only voc1 as the CONTEXT vocabulary. This prevents access to any other vocabularies unless an escape mechanism is included in the voc1 vocabulary. (Ragsdale) Delete all occurrences of ONLY from the search order. The effect is that only specified application vocabularies will be searched.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

```
: SEAL      ' >BODY CONTEXT #VOCS 2* ERASE CONTEXT ! ;
```

Example:

```
ONLY APPLICATION ALSO MENU SEAL
```

In this example, MENU and the APPLICATION will comprise the total search order.

Comment: This ideogram is a part of an alternative to the fig-Forth and 79S vocabulary structures. It was introduced by William F. Ragsdale at the 1982 FORML Conference. A number of implementations have been adopted.

Note: Any dictionary search will terminate when the link field address is given a value of zero. This may also be done in the fig-Forth and 79S vocabularies. It is useful in keeping end users restricted to a limited set of functions.

SEC

--- addr

MVP-FORTH DISK I-O

A variable used by the disk interface, containing the sector number last

SEC-READ

read or written relative to the last drive used. (*Fig*) Not defined. (*Fig - 8086/88 only*) A variable that contains the current sector number. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
VARIABLE SEC 0 SEC !
```

Fig: 8088/86:

```
0 VARIABLE SEC
```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
SEC @ U.
```

Get the address of this variable, then fetch its contents and print it.

Comment: Used together with TRACK in the calculations and setting up of the calls to the operating system.

SEC-READ

MVP-FORTH DISK I-O

Reads a disk sector into memory. All parameters must have been set by SET-DRIVE and SET-IO. The status on completion is stored in DISK-ERROR. (*Fig*) 8080 - Same as MVP. (*Fig - 8086/88 only*) The basic sector read routine (assumes SET-IO and SET-DRIVE have been executed). Called from R/W. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
HEX
```

```
: SEC-READ
```

```
DRIVE AX, MOV    USE BX, MOV 01 CX, MOV  
SEC DX, MOV     SI PUSH    BP PUSH    25 INT  
POPF    BP POP    SI POP    0 AH, MOV  
AX DISK-ERROR MOV    NEXT JMP    END-CODE
```

Fig: 8088/86:

```
CODE SEC-READ
```

```
GSEC CALL 0 AH, MOV AX DISK-ERROR+2, MOV NEXT JMP
```


F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: <R/W>   USE @ >R ROT USE ! SWAP MAX-DRV 0
DO I DR-DEN DENSITY !
  DUP BPDRV - -1
  IF BPDRV - I 1+ MAX-DRV =
    IF R> R> DDROP R> USE !
    1 ABORT" BLOCK OUT OF RANGE" THEN
  ELSE I DRIVE ! LEAVE
  THEN
LOOP SPBLK * SPBLK 0
DO DDUP T&SCALC
  IF SEC-READ
  ELSE SEC-WRITE
  THEN 1+ HDBT 4 - SPBLK / USE +!
LOOP
DDROP R> USE ! ;
```

The definition illustrates the use of this ideogram.

Comment: One of the basic disk read/write operators. With this it is possible to read any physical sector on a disk to any segment of memory. It is highly hardware dependent, but it belongs in a good Forth development system.

SEC-WRITE

MVP-FORTH DISK I-O

Writes a disk-sector from memory. All parameters must have been set by SET-DRIVE and SET-IO. The status on completion is stored in DISK-ERROR. Defined in: fig-Forth (8080) (*Fig - 8086/88 only*) The basic sector write routine (assumes SET-IO and SET-DRIVE have been executed). Called from R/W. (*Fig*) 8080 - Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

HEX

CODE SEC-WRITE

```
DRIVE AX, MOV   USE BX, MOV   01 CX MOV
SEC DX, MOV    SI PUSH   BP PUSH   26 INT
```

SEC/BLK

```
POPF    BP POP    SI POP    0 AH, MOV
AX DISK-ERROR MOV  NEXT JMP    END-CODE
```

Fig: 8088/86:

```
CODE SEC-WRITE    PSEC CALL    0 AH,    AX DISK-ERROR
MOV    NEXT JMP
```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: <R/W>    USE @ >R ROT USE ! SWAP MAX-DRV 0
DO  I DR-DEN DENSITY !
  DUP BPDRV - -1
  IF BPDRV - I 1+ MAX-DRV =
    IF R> R> DDROP R> USE !
      1 ABORT" BLOCK OUT OF RANGE" THEN
    ELSE I DRIVE ! LEAVE
  THEN
  LOOP SPBLK * SPBLK 0
  DO DDUP T&SCALC
    IF SEC-READ
    ELSE SEC-WRITE
    THEN 1+ HDBT 4 - SPBLK / USE +!
  LOOP
  DDROP R> USE ! ;
```

The definition illustrates the use of this ideogram.

Comment: One of the basic disk read/write operators. With this it is possible to write any sector on a disk from any location in memory. It is highly hardware dependent, but it belongs in a good Forth development system.

SEC/BLK

--- addr

MVP-FORTH DISK I-O

A variable beginning a seven item array containing the number of sectors on a drive of a give density format. (*Fig - 8088/86*) A constant equal to the number of sectors per block. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
VARIABLE SEC/BLK 2 SEC/BLK 1
      2 , 8 , 8 , 8 , 8 , 8 ,
```

Fig: 8088/86:

KBBUF/BPS CONSTANT SEC/BLK

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: SPBLK DENSITY @ 6 MIN 2* SEC/BLK + @ ;
```

The definition leaves the value of the sectors per block according to the present value of DENSITY.

Comment: Not all systems are formatted with 128 bytes per sector. The IBM Personal Computer under MDOS, formats 512 bytes per sector. Thus for the IBM implementation, the first value in the array was changed to 2.

SEC/DR

--- addr

MVP-FORTH DISK I-O

(8080 Implementation only) A variable beginning a seven item array containing the number of sectors on a drive of a given density format. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: SPDRV DENSITY @ 6 MIN 2 * SEC/DR + @ ;
```

The definition leaves the value of the sectors on a drive according to the present value of DENSITY.

SEE

Comment: An array used to calculate the actual drive being accessed according to the block number being requested. Some systems may require changing these values. If so, it will be necessary to poke the correct values in the proper places, and reselect your drive. Not used in MVP-FORTH 1.0405.03.

SEC/TR --- addr MVP-FORTH DISK I-O

A variable beginning a seven item array containing the number of sectors on each track on a drive of a given density format. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

HEX

VARIABLE SEC/TR 8 SEC/TR 1 8 , 1A ,
34 , 34 , 40 , 40 ,

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

: SPT DENSITY @ 6 MIN 2 * SEC/TR + @ ;

The definition leaves the number of sectors per track on a drive according to its current density.

Comment: An array used to calculate the actual drive being accessed according to the block number being requested. Some systems may require changing these values. If so, it will be necessary to poke the correct values in the proper places.

SEE | name --- NOT USED IN MVP-FORTH

(*Fig*) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) The user interface. To decompile something type SEE xxx.

*Implementations:***MVP: 8088/86:**

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: SEE
  BL WORD DUP C@
  IF ?UPPERCASE FIND 0= ?MISSING
  ELSE DROP
  THEN 1 ?ENOUGH (SEE) ;
```

F-PC:

```
: SEE ' (SEE) ;
```

Example:

SEE SEE

The example will display the compiled definition of SEE.

Comment: Several implementations of Forth use SEE to decompile the current implementation of the word following. It is a good ideogram to try when exploring new implementations of Forth. You might just find something.

SET-DRIVE

MVP-FORTH DISK I-O

(8080 Implementation only.) A system service call which makes subsequent disk reads and writes use the drive designated in DRIVE. T&SCALC is usually used to set the variable DRIVE and calls SET-DRIVE. Drive numbers range from 0 through MAX-DRV less one. (*Fig*) Not defined. (*Fig - 8086/88 only*) Sends the disk controller the new disk drive number. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

*Implementations:***MVP: 8088/86:**

Not used.

Fig: 8088/86:

```
CODE SET-DRIVE DRIVE CX, MOV SDSK CALL NEXT
JMP
```

F83: 8088/86:

SET-DRX

Not used.

F-PC:

Not used.

Example:

SET-DRIVE

Issue a command to the operating system to set a value for drive access according to the value presently in the variable DRIVE.

Comment: A utility which interlinks Forth to the CP/M operating system. It is not used in MVP-FORTH version 1.0405.03.

SET-DRX

n ---

MVP-FORTH DISK I-O

For drive number n, calculates and adds the necessary value to OFFSET. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

: SET-DRX

DR-DEN DENSITY ! BPDRV OFFSET +! ;

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

: DR1 DR0 0 SET-DRX ;

The definition calculates the necessary value for the variable OFFSET according to the present values established by CONFIGURE.

Comment: SET-DRX dynamically monitors changes in the value of MAX-DRV and the array DEN, but only when you execute it. When CONFIGURE is used to change the density on the various drives it is necessary to recalculate the values for OFFSET according to the selected drive.

SET-IO

MVP-FORTH DISK I-O

(8080 Implementation only.) A system service call which makes subsequent disk reads and writes use the drive last set by SET-DRIVE, not part of this ideogram. The memory address in variable USE, the sector number in SEC, and the track number in TRACK, must be set first. T&SCALC is usually used to set these variables. (*Fig*) Not defined. (*Fig - 8086/88 only*) Sets up the disk controller with Sector number, Track number and DMA address. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

CODE SET-IO

```

    USE CX, MOV    DSMAO CALL
    CS CX, MOV    SDMAS CALL    SEC CX, MOV    SSEC CALL
    TRACK CX, MOV  STRK CALL    NEXT JMP

```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```

: <R/W>
  USE @ >R ROT USE ! SWAP MAX-DRV 0
  DO I DR-DEN DENSITY !
    DUP BPDRV - -1
    IF BPDRV - I 1+ MAX-DRV =
      IF R> R> DDROP R> USE !
      1 ABORT" BLOCK OUT OF RANGE" THEN
    ELSE I DRIVE ! LEAVE
    THEN
  LOOP SPBLK * SPBLK 0
  DO DDUP T&SCALC
    IF SEC-READ
    ELSE SEC-WRITE
    THEN 1+ HDBT 4 - SPBLK / USE +!
  LOOP
  DDROP R> USE ! ;

```

SKIP

This example comes from the MVP-FORTH implementation.

Comment: A utility used to interface Forth with the operating system.

SIGN

n ---

MVP-FORTH

Insert the ASCII “-” (minus sign) into the pictured numeric output string, if n is negative. (*Fig*) Stores an ASCII “-” sign just before a converted numeric output string in the text output buffer when n is negative. n is discarded, but double number d is maintained. Must be used between <# and #>. (*79S*) Same as MVP. (*83S*) If n is negative, an ASCII “-” (minus sign) is appended to the pictured numeric output string. Typically used between <# and #>. (*F83*) If n1 is negative insert a minus sign into the string. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

```
HEX
: SIGN
  0<
  IF 2D HOLD THEN ;
```

Fig: 8088/86:

```
HEX
: SIGN
  ROT 0< IF 2D HOLD ENDIF ;
```

F83: 8088/86:

```
: SIGN
  0< IF ASCII - HOLD THEN ;
```

F-PC:

Same as F83.

Example:

```
-22. DUP ROT ROT DABS # #S ROT SIGN # TYPE
```

Enter a double precision negative value by ending with a decimal point and then format and type the value.

Comment: This ideogram will only function between <# and #>. *79S* flags SIGN as a “compile-only” ideogram; it is, however, generally conceded that was a typographical error.

SKIP

NOT USED IN MVP-FORTH

addr len char --- addr' len'

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Given the address and length of a string, and a character to look for, run through the string while we continue to find the character. Leave the address of the mismatch and the length of the remaining string. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
CODE SKIP
  AX POP  CX POP  $DONE JCXZ
  DI POP  DS DX MOV  DX ES MOV
  REPZ BYTE SCAS  0<>
  IF  CX INC  DI DEC  THEN
  DI PUSH  CX PUSH  NEXT
```

F-PC:

```
CODE SKIP
  POP AX  POP CX  JCXZ 0 $  POP DI
  MOV DX, ES  MOV ES, SSEG  REPZ
  MOV ES, DX  0<>
  IF  INC CX  DEC DI  THEN
  PUSH DI  PUSH CX  NEXT
  0 $:  PUSH CX  NEXT  END-CODE
```

Example:

```
PAD 20 EXPECT  AAABBBCCDDDD
PAD 20 ASCII A SKIP
```

The example places a series of characters at PAD. Setting the parameters and executing skip will leave the address of the first B and the length of 17 remaining characters on the stack.

Comment: The ideogram can be used in working with any array of ASCII characters. It is a primitive in the development of string handling functions.

SMUDGE

MVP-FORTH

Used during word definition to toggle the "smudge bit" in a definition's

SP!

name field. This prevents an uncompleted definition from being found during dictionary searches, until compiling is completed without error. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
HEX
: SMUDGE
  LATEST 20 TOGGLE ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: ; ?CSP COMPILE EXIT SMUDGE [COMPILE] [ ;
```

This example comes from the MVP-FORTH implementation.

Comment: The closing smudge in ";" cancels out the initial smudge performed by ":". If, because of an error, a colon definition is terminated before completion, its smudge bit will hide it from any dictionary searches. Although you will see it in your VLIST, trying to FORGET it will return an error with the message "NOT IN CURRENT VOCABULARY". If the offending ideogram is at the top of the dictionary, executing SMUDGE will expose it for removal.

SP!

MVP-FORTH

A computer dependent procedure to initialize the stack pointer from S0. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) (Warning, this is different from *fig-Forth*) Sets the parameter stack pointer to the specified value. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

```
CODE SP!
  UP BX, MOV [BX] 6 + SP, MOV NEXT JMP END-CODE
```

Fig: 8088/86:

Same as MVP.

```

F83: 8088/86:
      CODE SP!
      SP POP    NEXT

```

F-PC:
Same as F83.

Example:

```
: ABORT  SP! ?STACK  [COMPILE]
FORTH DEFINITIONS  QUIT  ;
```

The definition is used in conjunction with an error routine to reset the stack to its empty position.

Comment: Only the data stack is cleared. The return stack and everything else remain intact.

```
SP0      --- addr      MVP-FORTH
```

A user variable that contains the initial value of the stack pointer. (See s0). (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Empty parameter stack for this task. (F-PC) Same as F83.

Pronounced: s-p-zero

Implementations:

MVP: 8088/86:
HEX 06 USER SP0

Fig: 8088/86:
Not used.

F83: 8088/86:
VARIABLE SP0

F-PC:
Same as F83.

Example:

```
: S0    SP0    @    ;
```

The example is from the MVP-FORTH source code.

Comment: Although this user variable is given a different name in *fig-Forth* and *Starting Forth*, its MVP-FORTH identifier makes the ideogram S0 available to function according to the *79S Reference Word Set*.

SPACE

SP@ --- addr

MVP-FORTH

Return the address of the top of the stack, just before SP@ was executed. (Fig) A computer dependent procedure to return the address of the stack position to the top of the stack, as it was before SP@ was executed. (e.g. 1 2 SP@ @ . . . would type 2 2 1) (79S Reference Word Set) Same as MVP. (83S) Not defined. (F83) Return the address of the next entry on the parameter stack (F-PC) Same as F83.

Pronounced: s-p-fetch

Implementations:

MVP: 8088/86:

```
CODE SP@
    SP AX, MOV    APUSH JMP    END-CODE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

CODE SP@ SP AX MOV 1PUSH

F-PC:

Same as F83.

Example:

```
: DEPTH  SP@  S0  SWAP  -  2  /  ;
```

This example is taken from the MVP-FORTH implementation.

Comment: The 79S prefers you to use the ideogram DEPTH in place of the implementation dependent SP@.

SPACE ---

MVP-FORTH

Transmit an ASCII blank to the current output device. (Fig) Transmit an ASCII blank to the output device. (79S) Same as MVP. (83S) Displays an ASCII space. (F83) Send a space to the terminal (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

```
: SPACE BL EMIT ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Same as MVP.

F-PC:

Same as MVP.

Example:

```
333 3 .R SPACE 444 3 .R
```

This example illustrates the formatted output of two 3-digit numbers separated by one space.

Comment: This ideogram will place the proper ASCII value regardless of the current number base.

SPACES

n ---

MVP-FORTH

Transmit n ASCII spaces to the current output device. Take no action for n of zero or less. (*Fig*) Transmit n ASCII blanks to the output device. (*79S*) Same as MVP. (*83S*) Displays +n ASCII spaces. Nothing is displayed if +n is zero. (*F83*) Send a set of spaces to the terminal (*F-PC*) Send a set of spaces to the terminal.

Implementations:

MVP: 8088/86:

```
: SPACES
  0 MAX ?DUP
  IF 0
    DO SPACE LOOP
  THEN ;
```

Fig: 8088/86:

```
: SPACES
  0 MAX -DUP
  IF 0
    DO SPACE LOOP
  ENDIF ;
```

F83: 8088/86:

```
: SPACES
  0 MAX 0 ?DO SPACE LOOP ;
```

F-PC:

```
: SPACES
  0MAX DUP 80 <
  IF SPCS SWAP TYPE
  ELSE 80 /MOD 0
```

SPBLK

?DO SPCS 80 TYPE LOOP SPCS SWAP TYPE
THEN

Example:

33 4 .R 20 SPACES 44 4 .R

This example illustrates the formatted output of two 4-digit numeric fields separated by twenty spaces.

Comment: The usefulness of this somewhat trivial ideogram justifies its inclusion in 79S and MVP-FORTH.

SPAN

--- addr

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) The address of a variable containing the count of characters actually received and stored by the last execution of EXPECT. See: EXPECT (F83) Number of characters input by EXPECT. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

VARIABLE SPAN

F-PC:

Same as F83.

Example:

PAD 20 EXPECT AAABBBCCC
SCAN @ .

Input a number of characters to PAD. Then find the number and print it.

Comment: A function which can be used as a primitive in developing a family of string functions.

SPBLK

--- n

MVP-FORTH DISK I-O

Find the value in the array SEC/BLK according to the value of DENSITY. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

*Implementations:***MVP: 8088/86:**

```
: SPBLK
  DENSITY @ 6 MIN 2* SEC/BLK + @ ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: SPDRV
  BPDRV SPBLK * ;
```

The example is taken from the source code.

Comment: The ideogram is used to accommodate drives formatted with various numbers of sectors per block rather than the 8 sectors per block used in most systems; 2 sectors per block in DOS.

SPDRV

--- n

MVP-FORTH DISK I-O

Find the value in the array SEC/DR according to the value of DENSITY.
(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

*Implementations:***MVP: 8088/86:**

```
: SPDRV
  BPDRV SPBLK * ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
SPDRV .
```

Get the number of sectors on a drive according the the present value

SPT

of DENSITY and print it.

Comment: A factored Forth utility used in setting up for disk access.

SPLIT n1 --- n2 n3 NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Split the 16-bit value n1 into two stack entries which are the low and hi bytes of n1. The hi byte is on the top of the stack.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

CODE SPLIT

```
POP BX      SUB AX, AX      MOV AL, BL
PUSH AX     MOV AL, BH      1PUSH  END-CODE
```

Example:

```
HEX 3AC0 SPLIT . . DECIMAL
```

Split the hexadecimal integer 3AC0 into the low and hi bytes, then print them.

Comment: The ideogram provides a quick way for unpacking byte characters.

SPT --- n MVP-FORTH DISK I-O

Find the value in the array SEC/TR according to the value of DENSITY. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: SPT  DENSITY  @  6  MIN  2*  SEC/TR  +  @  ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

SPT .

Get the number of sectors per track according to the value present value of DENSITY and print it.

Comment: A factored Forth utility used in setting up for disk access.

STATE

--- addr

MVP-FORTH

Leave the address of the variable containing the compilation state. A non-zero content indicates compilation is occurring, but the value itself may be installation dependent. (*Fig*) A user variable containing the compilation state. A non-zero value indicates compilation. The value itself may be implementation dependent. (*79S*) Same as MVP. (*83S*) The address of a variable containing the compilation state. A non-zero content indicates compilation is occurring, but the value itself is system dependent. A Standard Program may not modify this variable. (*F83*) Not defined. (*F-PC*) A variable that holds the compilation state. If the contents of STATE are 0, then the Forth system is in interpret mode. Otherwise the system is in compilation mode.

Implementations:

MVP: 8088/86:

HEX
50 USER STATE

Fig: 8088/86:

HEX
24 USER STATE

F83: 8088/86:

VARIABLE STATE

F-PC:

Same as F83.

Example:

:] C0 STATE ! ;

This example comes from the MVP-FORTH implementation.

Comment: Although STATE is widely used and is part of the 79S, it is

SYSCALL

not indispensable. *Starting Forth* seems to manage quite well without it.

SWAP

n1 n2 --- n2 n1

MVP-FORTH

Exchange the top two stack values. (*Fig*) Exchange the top two values on the stack. (*79S*) Same as MVP. (*83S*) The top two stack entries are exchanged. (*F83*) Exchange the top two elements on the stack. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

CODE SWAP

DX POP AX POP DPUSH JMP END-CODE

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

CODE SWAP

DX POP AX POP 2PUSH

F-PC:

Same as F83.

Example:

33 44 SWAP

Enter two values on the stack and then exchange them.

Comment: This ideogram is one of the fundamental stack operators.

SYSCALL

n1 n2 --- n3

MVP-FORTH

Setup and execute system function calls. n1 is the function code number and n2 is the parameter value to be placed in the DE register. The BDOS is then called and the error code in register A, if any, is placed on the stack. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

HEX

: SYSCALL

SWAP 100 * SWAP 0 0 ROT 21 INTCALL ;

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
2 65  SYSCALL  DROP
```

The character A will be printed at the terminal and, since this function returns no error, a meaningless value returned on the stack is dropped.

Comment: This ideogram gives the user access to most of the system's functions. It is used to implement SAVE-FORTH.

T&SCALC

n ---

MVP-FORTH DISK I-O

Track, sector and drive calculations for disk input and output . n is the total sector displacement from the first logical drive to the desired sector. The corresponding drive, track, and sector number are calculated. If the drive number is different from the contents of DRIVE, the new drive number is stored in DRIVE and SET-DRIVE is executed. The track number is stored in TRACK; the sector number is stored in SEC. T&SCALC is usually executed before SET-DRIVE. The function is implementation dependent. (Fig) Not defined. (Fig - 8086/88 only) Calculates: Drive, Track and Sector values from a sector displacement number "n". The results are stored in the appropriate variables. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

```
: T&SCALC  'T&SCALC  @  EXECUTE  ;
```

Fig: 8088/86:

```
(DOUBLE DENSITY)
```

```
: T&SCALC  DENSITY @
```

```
IF  SODRV2 ( System value )
```

```
  /MOD MAX-DRV MIN  DUP DRIVE @ =
```

```
  IF DROP ELSE DRIVE ! SET-DRIVE  ENDIF
```

```
  SPT2 ( System value ) /MOD TRACK ! 1+ SEC ! ;S
```

```
ELSE  SPDRV1 ( System value )
```

```
  /MOD MAX-DRV MIN  DUP DRIVE @ =
```

```
  IF DROP ELSE DRIVE ! SET-DRIVE  ENDIF
```

TASK

```
SPT1 ( System value )/MOD TRACK ! 1+ SEC !  
ENDIF;
```

F83: 8088/86:
Not used.

F-PC:
Not used.

Example:

```
: <R/W>  
USE @ >R ROT USE ! SWAP MAX-DRV 0  
DO I DR-DEN DENSITY !  
  DUP BPDRV - -1  
  IF BPDRV - I 1+ MAX-DRV =  
    IF R> R> DDROP R> USE !  
    1 ABORT" BLOCK OUT OF RANGE" THEN  
  ELSE I DRIVE ! LEAVE  
  THEN  
LOOP SPBLK * SPBLK 0  
DO DDUP T&SCALC  
  IF SEC-READ  
  ELSE SEC-WRITE  
  THEN 1+ HDBT 4 - SPBLK / USE +!  
LOOP  
DDROP R> USE ! ;
```

This example comes from the MVP-FORTH implementation.

Comment: This is a revised implementation from that in fig-Forth; it takes into account the number of sectors which are present on each disk in making the calculation. If T&SCALC cannot map n onto a physical sector, no error message is given and the values of DRIVE, TRACK, and SEC are not altered. In MVP-FORTH, this ideogram is vectored for the convenience of the programmer, defaulting to <T&SCALC>.

TASK

NOT USED IN MVP-FORTH

(Fig) A no-operation word which can mark the boundary between applications. By forgetting TASK and re-compiling, an application can be discarded in its entirety. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

FORGET TASK : TASK ;

The example first forgets the ideograms in the dictionary through TASK, frequently the end of the boot up version, and then replaces it.

Comment: Regardless of the ideogram you use for this purpose, remember to redefine it after forgetting it. The name you choose is secondary to the technique. TASK is simply a dummy placeholder marking a certain point in the dictionary.

TEXT

c ---

MVP-FORTH

Accept characters from the input stream, as for WORD, into PAD, blank filling the remainder of PAD to 64 characters. (*Fig*) Not defined. (*79S Reference Word Set*) Same as MVP. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

: TEXT

HERE C/L 1+ BLANK WORD BL OVER

DUP C@ + 1+ C! PAD C/L 1+ CMOVE ;

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

THRU

Example:

```
: DELIMIT    BL    TEXT    ;
```

DELIMIT, when executed, will take the next word from the input stream and place it with a preceding length byte, at PAD.

Comment: Though this ideogram is usually present only in the EDITOR vocabulary, it can be convenient to have it available along with a few others in Forth when the EDITOR vocabulary is not loaded.

THEN

MVP-FORTH

Used in a colon-definition. THEN is the point where execution resumes after ELSE or IF (when no ELSE is present). (*Fig*) An alias for ENDIF. (79S) Same as MVP. (83S) THEN is the point where execution continues after ELSE, or IF when no ELSE is present. sys is balanced with its corresponding IF or ELSE. See: IF ELSE (F83) Not defined. (F-PC) Terminate a branch structure. Used in the form: flag IF ... ELSE ... THEN.

Implementations:

MVP: 8088/86:

```
: THEN    ?COMP    2    ?PAIRS    HERE    OVER
-    SWAP    !    ;    IMMEDIATE
```

Fig: 8088/86:

```
: THEN [COMPILE] ENDIF ; IMMEDIATE
```

F83: 8088/86:

```
: THEN    1000 ?PAIRS >RESOLVE ; IMMEDIATE
```

F-PC:

```
: THEN    COMPILE DOTHEN ?>RESOLVE ; IMMEDIATE
```

Example:

```
: TEST 1 IF ." ONE " THEN ;
```

The definition will always print "ONE".

Comment: ENDIF, an obsolete alias for THEN, may appear in older programs.

THRU

n1 n2 ---

MVP-FORTH UTILITY

Load consecutively the blocks from n1 through n2. (*Fig*) Not defined. (79S *Reference Word Set*) Same as MVP. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

*Implementations:***MVP: 8088/86:**

```
: THRU 1+ SWAP
DO I U. I LOAD
?TERMINAL IF LEAVE THEN
LOOP ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: THRU
2 ?ENOUGH 1+ SWAP
?DO I LOAD LOOP ;
```

F-PC:

Not used.

Example:

```
10 20 THRU
```

This example will load screens 10 through 20, printing the number of each screen as it is loaded.

Comment: This ideogram is the preferred alternative to "-->" for loading a contiguous series of screens. By typing each screen's number as it is loading, this version lets one monitor the progress of a lengthy compilation.

TIB --- addr**MVP-FORTH**

A user variable containing the address of the terminal input buffer. (*Fig*) Same as MVP. (*79S*) Not defined. (*83S*) The address of the text input buffer. This buffer is used to hold characters when the input stream is coming from the current input device. The minimum capacity of TIB is 80 characters. (*F83*) Leaves address of text input buffer. (*F-PC*) Same as F83.

*Implementations:***MVP: 8088/86:**

```
HEX
0A USER TIB
```

Fig: 8088/86:

Same as MVP.

TITLE

F83: 8088/86:

: TIB 'TIB @ ;

F-PC:

CODE TIB PUSH 'TIB NEXT END-CODE

Example:

TIB @ U.

Get the address of this user variable, fetch its contents and print it unsigned.

Comment: Although 79S requires a terminal input buffer, the means of locating it is left up to the implementation. The MVP-FORTH implementation uses the fig-Forth approach.

TITLE

MVP-FORTH UTILITY

Print a fixed message, "MOUNTAIN VIEW PRESS FORTH VERSION 1.00.03", followed by a carriage return. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

: TITLE

CR 10 SPACES ." MOUNTAIN VIEW PRESS FORTH"
." VERSION 1.0405.03 " CR ;

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

FIND TITLE 'TITLE !

This example resets TRIAD to print its default message.

Comment: This default message is printed at the bottom of each page by TRIAD using the vector 'TITLE. Any other ideogram may be defined and used to replace TITLE. The ideogram then vectored by 'TITLE will be invoked by TRIAD.

TO n --- NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined (F83) You can use TO instead of HOP if you know the destination screen number instead of the number of screens to skip. (F-PC) Not defined. (Bartholdi) The operator sets a variable %VAR to 1, forcing the next-called variable [usually VALUE] to store the content on top of the stack in its parameter field instead of fetching it.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

: TO

SWAP BL WORD NUMBER DROP OVER - HOP SWAP ;

F-PC:

Not used.

Bartholdi:

: TO 1 %VAR ! ;

Example:

1234 TO COUNTER

COUNTER has been defined with VALUE such that 1234 is then stored in it.

Comment: The ideogram must be used in conjunction with variables defined with an ideogram implemented to make use of the function. VALUE is the name of such a defining word. Bartholdi illustrates how it may also be used with defined ARRAYS and other structures. The function of this ideogram in F83 is not in common usage but may be confused.

TOGGLE addr b --- MVP-FORTH

Complement the contents of addr by the bit pattern b. (Fig) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

TOS

CODE TOGGLE
AX POP BX POP AL [BX], XOR NEXT JMP END-CODE

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: SMUDGE LATEST 20 TOGGLE ;
```

The definition, taken from the MVP-FORTH implementation, utilizes this ideogram to complement the value of bit 6 in the byte at LATEST.

Comment: Only the byte at addr is affected.

TOS

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Saved during Task switching. (F-PC) Top Of Stack, saved during Task switching.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

VARIABLE TOS

F-PC:

Same as F83.

Example:

```
: TASK:  
  CREATE TOS HERE #USER @ SMOVE  
  HERE ENTRY LOCAL LINK !  
  DUP HERE + DUP RP0 ! 100 - SP0 ! SWAP UP !  
  HERE #USER @ + HERE DP LOCAL !  
  HERE SLEEP ALLOT ;
```

The example is a defining function as defined by Henry Laxen.

Note: Not all of the words are in common usage. Refer to Henry Laxen's Multi-Tasking, Part I and Part II in *Forth Dimensions* Vol. V, Nos. 4 & 5, 1983/1984

Comment: Depending upon the application, multi-tasking can be accomplished in a number of ways.

TRACK --- addr MVP-FORTH DISK I-O

A variable used by disk I-O. Contains the track number last read or written relative to the current drive. (*Fig*) Not defined. (*Fig - 8086/88 only*) A variable that contains the current track number. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

VARIABLE TRACK 0 TRACK !

Fig: 8088/86:

0 VARIABLE TRACK

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

TRACK @ .

Get the address of the variable, fetch its contents and print it.

Comment: A variable used in interfacing Forth with system.

TRAVERSE addr1 n --- addr2 MVP-FORTH

Move across the name field of a fig-Forth variable length name field. addr1 is the address of either the length byte or the last letter. If n = 1, the motion is toward high memory; if n = -1, the motion is toward low memory. The addr2 resulting is the address of the other end of the name. (*Fig*) Same as MVP. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Run through a name field in the specified direction. Terminate when a byte whose high order bit is on is detected. (*F-PC*) Same as F83.

TRIAD

Implementations:

MVP: 8088/86:

```
: TRAVERSE  SWAP
  BEGIN OVER + 07F OVER C@ < UNTIL
  SWAP DROP ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
CODE TRAVERSE
  CX POP    BX POP    CX BX ADD
  BEGIN    0 [BX] AL MOV    128 # AL AND    0=
  WHILE    CX BX ADD
  REPEAT    BX PUSH    NEXT
```

F-PC:

```
CODE TRAVERSE
  POP CX    POP BX    ADD BX, CX
  PUSH ES   MOV ES, YSEG
  BEGIN    MOV ES: AL, 0 [BX]    AND AL, # 128    0=
  WHILE    ADD BX, CX
  REPEAT
  POP ES    PUSH BX    NEXT    END-CODE
```

Example:

```
: NFA    5    -    -1    TRAVERSE    ;
```

The definition is used to find the name field address, given the parameter field address.

Comment: This ideogram is necessary in implementations of Forth with fig-Forth style name headers. However, 79S prohibits such poking around with the header and the operation would remain headerless in a "pure" implementation. On the other hand, a good Forth development system should have it available.

TRIAD

scr ---

MVP-FORTH UTILITY

Display on the selected output device the three screens which include that numbered scr, beginning with a screen evenly divisible by three. Output is suitable for source text records and includes an alterable bottom title. (*Fig*) Same as MVP but with the title being taken from line 15 of screen 4.. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```

: TRIAD
  PAGE 0 3 U/MOD SWAP DROP
  3 * 3 OVER + SWAP
  DO CR I LIST ?TERMINAL IF LEAVE THEN 1
  /LOOP 'TITLE @ EXECUTE ;

```

Fig: 8088/86:

```

HEX
: TRIAD
  00FF EMIT 3 / 3 * 3 OVER + SWAP
  DO CR I LIST ?TERMINAL IF LEAVE ENDIF
  LOOP CR 15 MESSAGE CR ;

```

F83: 8088/86:

```

: TRIAD
  12 EMIT 3 / 3 * 3 BOUNDS
  DO I LIST LOOP :

```

F-PC:

Not used.

Example:

31 TRIAD

The example will print three screens beginning with screen 30.

Comment: By making each page evenly divisible by three, screen listings may be maintained in a loose-leaf notebook. This version of TRIAD prints a vectored bottom title. Substituting a different title text is possible by defining a new message routine and storing its compilation address in 'TITLE.

TRUE

--- f

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Constant for clarity. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

TYPE

F83: 8088/86:

-1 CONSTANT TRUE

F-PC:

Same as F83.

Example:

TRUE

Place a true flag on the stack.

Comment: Gives a name to the value for true. It serves as a parallel with FALSE and the actual values can be changed without going through all of the program.

TUCK

NOT USED IN MVP-FORTH

n1 n2 --- n2 n1 n2

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Tuck the first element under the second one. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE TUCK

AX POP DX POP AX PUSH 2PUSH

F-PC:

Same as F83.

Example:

1 2 TUCK .S

The example tests the function. The result is displayed.

Comment: The ideogram is in common usage. In appropriate applications it is convenient. There are several ways it can be defined in high level Forth: : TUCK DUP ROT ROT ; or : TUCK SWAP OVER ;

TYPE

addr n ---

MVP-FORTH

Transmit n characters beginning at address to the current output device. No action takes place for n less than or equal to zero. (Fig)

Transmit count characters from addr to the selected output device. (79S) Same as MVP. (83S) +n characters are displayed from memory beginning with the character at addr and continuing through consecutive addresses. Nothing is displayed if +n is zero. See: "9.5.4 TYPE" (F83) Display the given string on the terminal. (F-PC) A deferred word used to Print a string to the current output device from the segment specified in the variable TYPESEG.

Implementations:

MVP: 8088/86:

```
: TYPE
  DUP 0>
  IF OVER + SWAP
    DO I C@ EMIT 1 /LOOP
  ELSE DDROP THEN ;
```

Fig: 8088/86:

```
: TYPE
  DUP
  IF OVER + SWAP
    DO I C@ EMIT LOOP
  ELSE DROP THEN ;
```

F83: 8088/86:

DEFER TYPE (A coded implementation is vectored.)

F-PC:

Same as F83.

Example:

```
PAD COUNT TYPE
```

Get the address of PAD, then get the length of the text at PAD and advance the address by 1, and then type out the contents.

Comment: The definitions vary slightly but for most work they function identically. Note that you should expect trouble if your implementation will not handle fields crossing addresses 0 or decimal 32768, that is the transition from positive to negative signed integers.

U*

un1 un2 --- ud3

MVP-FORTH

Perform an unsigned multiplication of un1 by un2, leaving the double number product ud3. All values are unsigned. (Fig) Leave the unsigned double number product of two unsigned numbers. (79S) Same as MVP. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

U.

Pronounced: u-times

Implementations:

MVP: 8088/86:

CODE U*

AX POP BX POP BX MUL DX AX, XCHG
DPUSH JMP END-CODE

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

333 444 U* <# #S #> TYPE

Enter two unsigned single precision values, return the 32-bit unsigned product to the stack and print it.

Comment: Allows working with unsigned numbers and provides full precision in the double precision result. No data is lost and no overflow is possible. U* can be the basis for all other multiplications. It has the same function as UM* in some dialects.

U.

un ---

MVP-FORTH

Display un converted according to BASE as an unsigned number, in a free-field format, with one trailing blank. (*Fig*) Not defined. (*79S*) Same as MVP. (*83S*) u is displayed as an unsigned number in a free-field format. (*F83*) Output as an unsigned single number with trailing space. (*F-PC*) Same as F83.

Pronounced: u-dot

Implementations:

MVP: 8088/86:

: U. 0 D. ;

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

: U. (U.) TYPE SPACE ;

F-PC:

Same as F83.

Example:

HEX BCAD DECIMAL U.

Enter a hex value greater than 8000, return to decimal and print it unsigned. If simply . were used, the value printed would be negative.

Comment: This ideogram permits the output of an address which might otherwise appear as a negative number. It is defined in many implementations of Forth including fig-Forth even though it is not included in the *Installation Manual*.

U.R

un1 n2 ---

MVP-FORTH SUPPLEMENTAL

Output un1 as an unsigned number right justified in a field n2 characters wide. If n2 is smaller than the characters required for n1, no leading spaces are given. (*Fig*) Not defined. (*79S Reference Word Set*) Same as MVP. (*83S*) Not defined. (*F83*) Output as an unsigned single number right justified. (*F-PC*) Same as F83.

Pronounced: u-dot-r

Implementations:

MVP: 8088/86:

: U.R 0 SWAP D.R ;

Fig: 8088/86:

Not used.

F83: 8088/86:

: U.R >R (U.) R> OVER - SPACES TYPE ;

F-PC:

Same as F83.

Example:

HEX BCDA DECIMAL 6 U.R

Enter a hex value greater than 8000, return to decimal and print the value in a field right justified to six spaces.

Comment: This allows the output of addresses or block numbers in a formatted field.

U/MOD

U/ ud u1 --- u2 u3 NOT USED IN MVP-FORTH

(Fig) Leave the unsigned remainder u2 and unsigned quotient u3 from the unsigned double dividend ud and unsigned divisor u1 . (Fig 8088/86 Modification) Same as the fig model except returns a -1 for both the quotient and remainder (u2 and u3) when the divisor is zero. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Implementations:

MVP: 8088/86: Could be defined as:

```
: U/  
  U/MOD ;
```

Fig: 8088/86:

```
CODE U/  
  BX POP    DX POP    AX POP  
  BX DX, CMP    DZERO JNB    BX DIV    DPUSH JMP  
  HERE LABEL DZERO  
  -1 AX, MOV    AX DX, MOV    DPUSH JMP
```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
45000.    36000    U/
```

Divide the double dividend by the single divisor, printing the quotient and remainder. U/ treats all quantities as unsigned. A signed division would return different results for the same inputs.

Comment: This obsolete synonym for U/MOD may appear in older programs.

U/MOD ud1 un2 --- un3 un4 MVP-FORTH

Perform the unsigned division of double number ud1 by un2, leaving the remainder un3, and quotient un4. All values are unsigned. (Fig) Not defined. (79S) Same as MVP. (83S) Not defined. (F83) Not defined. (F-PC) Not defined.

Pronounced: u-divide-mod

Implementations:

MVP: 8088/86:

CODE U/MOD

```

    BX POP    DX POP    AX POP
    BX DIV    DPUSH JMP  END-CODE

```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```

45000. 36000  U/MOD  .  .

```

Divide the double dividend by the single divisor, printing the quotient and remainder. U/MOD treats all quantities as unsigned. A signed division such as M/ would return different results for the same inputs.

Comment: All division, signed and unsigned, may be expressed in terms of U/MOD. This function has been named UM/MOD in some dialects.

U< un1 un2 --- flag MVP-FORTH

Leave the flag representing the magnitude comparison of $un1 < un2$ where un1 and un2 are treated as 16-bit unsigned integers. (*Fig 8088/86*) Leaves a true flag if "u1" is less than "u2"; otherwise leaves a false flag. (This is an unsigned comparison.) (*79S*) Same as MVP. (*83S*) flag is true is (sic) u1 is less than u2. (*F83*) Compare the top two elements on the stack as unsigned integers and return true if the second is less than the first. Be sure to use U< whenever comparing addresses, or else strange things will happen beyond 32K. (*F-PC*) If unsigned n1 is less than unsigned n2, return TRUE, otherwise FALSE.

Pronounced: u-less-than

Implementations:

MVP: 8088/86:

```

: U<
  0 SWAP 0 D< ;

```

Fig: 8088/86:

```

: U<
  2DUP XOR 0<

```

UM*

IF DROP 0< 0= ELSE - 0< THEN ;

F83: 8088/86:

CODE U<

AX POP BX POP AX BX CMP YES JB
NO #) JMP END-CODE

F-PC:

CODE U<

POP CX POP AX SUB AX, CX SBB AX, AX
1PUSH END-CODE

Example:

45000 35000 U<

Enter two large values which if signed would be negative, and compare them leaving a flag on the stack, in this case a 0 for false.

Comment: This ideogram is useful when dealing with addresses which may otherwise confuse negative values.

UM*

n1 n2 --- d

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) ud is the unsigned product of u1 times u2. All values and arithmetic are unsigned. (F83) Return a 32-bit unsigned product of two 16-bit unsigned numbers. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE UM*

AX POP BX POP BX MUL DX AX XCHG 2PUSH

F-PC:

Same as F83.

Example:

333 444 UM* <# #S #> TYPE

Enter two unsigned single precision values, return the 32-bit unsigned product to the stack, format and print it.

Comment: Allows working with unsigned numbers and provides full precision in the double precision result. No data is lost and no overflow is possible. UM* can be the basis for all other multiplications. It has the same function as U* in fig-Forth, 79S, and MVP-FORTH.

UM/MOD

NOT USED IN MVP-FORTH

ud un --- urem uquot

(Fig) Not defined. (79S) Not defined. (83S) u2 is the remainder and u3 is the floor of the quotient after dividing ud by the divisor u1. All values and arithmetic are unsigned. An error condition results if the divisor is zero or if the quotient lies outside the range {0..65,535}. See: "floor, arithmetic". (F83) This is the division primitive in Forth. All other division operations are derived from it. It takes a double number, d1, and divides by a single number n1. It leaves a remainder and a quotient on the stack. For a clearer understanding of arithmetic consult Knuth Volume 2 on *Seminumerical Algorithms*. (F-PC) The unsigned double numerator ud is divided by an unsigned single denominator un to produce an unsigned quotient and unsigned remainder. The quotient is at the top of the stack.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

CODE UM/MOD

BX POP DX POP AX POP BX DX CMP

U>= (divide by zero?)

IF -1 # AX MOV AX DX MOV 2PUSH, THEN

BX DIV 2PUSH

F-PC:

Same as F83.

Example:

45000. 36000 UM/MOD

Divide the double dividend by the single divisor, printing the quotient and remainder. UM/MOD treats all quantities as unsigned. A signed division such as UM/ would return different results from the same inputs.

UMIN

Comment: All division, signed and unsigned, may be expressed in terms of UM/MOD. Replaces U/MOD in fig-Forth, 79S, and MVP-FORTH

UMAX

NOT USED IN MVP-FORTH

un1 un2 --- un3

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Return the maximum of n1 and n2, treated as unsigned numbers.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: UMAX 2DUP U< IF SWAP THEN DROP ;
```

F-PC:

```
CODE UMAX
```

```
POP AX      POP BX      CMP BX, AX      U<=  
IF 1PUSH THEN  
PUSH BX     NEXT      END-CODE
```

Example:

```
50000 20000 UMAX U.
```

Enter the two values (Note: 50000 is the unsigned value of -15536) and perform the function. The unsigned output of U. is necessary to display the correct result.

Comment: There are many cases where unsigned integers are used and appropriate functions are necessary. F-PC has implemented the function in assembly language which should be faster than the high level implementation of F83.

UMIN

NOT USED IN MVP-FORTH

un1 un2 --- un3

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Return the minimum of n1 and n2, treated as unsigned numbers.

Implementations:
MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: UMIN      2DUP U< IF  SWAP  THEN  DROP  ;
```

F-PC:

```
CODE UMIN
  POP AX      POP BX      CMP BX, AX      U<=
  IF  PUSH BX,  NEXT
  THEN  1PUSH    END-CODE
```

Example:

```
50000 20000  UMIN  U.
```

Enter the two values (Note: 50000 is the unsigned value of -15536) and perform the function. The unsigned output of U. is necessary to display the correct result.

Comment: There are many cases where unsigned integers are used and appropriate functions are necessary. F-PC has implemented the function in assembly language which should be faster than the high level implementation of F83.

UNTIL
MVP-FORTH

```
addr n ---  ( compiling )
fl  ---  ( executing )
```

Within a colon-definition, mark the end of a BEGIN-UNTIL loop, which will terminate based on a flag. If flag is true, the loop is terminated. If flag is false, execution returns to the first word after BEGIN. BEGIN-UNTIL structures may be nested. (*Fig*) Occurs within a colon-definition in the form BEGIN ... UNTIL. At run-time, UNTIL controls the conditional branch back to the corresponding BEGIN. If f is false, execution returns to just after BEGIN; if true, execution continues ahead. At compile-time, UNTIL compiles (0BRANCH) and an offset from HERE to addr. n is used for error tests. (79S) Same as MVP. (83S) Used in the form: BEGIN ... flag UNTIL. Marks the end of a BEGIN-UNTIL loop which will terminate based on flag. If flag is true, the loop is terminated. If flag is false, execution continues to just after the corresponding BEGIN. sys is balanced with its corresponding

UP

BEGIN. See: BEGIN (F83) Not defined. (F-PC) Marks end of a BEGIN ... UNTIL loop; terminate if flag f1 is true.

Implementations:

MVP: 8088/86:

```
: UNTIL
  1 ?PAIRS
  COMPILE 0BRANCH HERE - , ; IMMEDIATE
```

Fig: 8088/86:

```
: UNTIL
  1 ?PAIRS
  COMPILE 0BRANCH BACK ; IMMEDIATE
```

F83: 8088/86:

```
: UNTIL
  COMPILE ?BRANCH 2000 ?PAIRS <RESOLVE ; IMMEDIATE
```

F-PC:

```
: UNTIL
  COMPILE ?UNTIL ?<RESOLVE ; IMMEDIATE
```

Example:

```
: TEST -1
  BEGIN 1+ DUP . DUP 9 =
  UNTIL DROP ;
```

The definition will print the digits 0 through 9.

Comment: Older versions of Forth use the obsolete synonym, END.

UP

--- addr

MVP-FORTH

A constant returning a pointer to the cell in low memory which holds the pointer to the user area. (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Holds a pointer to the current user area. (multitasking) (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

```
UP CONSTANT UP
( The cross-compiler uses the label UP to define
  the constant UP. )
```

Fig: 8088/86:

Not used.

F83: 8088/86:
 VARIABLE UP

F-PC:
 Same as F83.

Example:

```
' SP0 @ UP @ + @ U.
```

This contrived example will print the value in the user variable SP0.

Comment: Multi-user Forth systems maintain a separate set of user variables for each terminal task. Switching between tasks involves saving and restoring CPU registers and selecting a new user area. In 8080/Z80 systems, a user area is activated by storing its starting address into the cell at UP.

UPDATE

MVP-FORTH

Mark the most recently referenced block as modified. The block will subsequently be automatically transferred to mass storage should its memory buffer be needed for storage of a different block, or upon execution of SAVE-BUFFERS. (*Fig*) Marks the most recently referenced block (pointed to by PREV) as altered. The block will subsequently be transferred automatically to disc should its buffer be required for storage of a different block. (*79S*) Same as MVP. (*83S*) The currently valid block buffer is marked as modified. Blocks marked as modified will subsequently be automatically transferred to mass storage should its memory buffer be needed for storage of a different block or upon execution of FLUSH or SAVE-BUFFERS. (*F83*) Mark the most recently used buffer as modified. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:
 HEX
 : UPDATE
 PREV @ @ 8000 OR PREV @ ! ;

Fig: 8088/86:
 Same as MVP.

F83: 8088/86:
 : UPDATE
 >UPDATE ON ;

F-PC:

USE

Not used.

Example:

```
50 BLOCK UPDATE
```

Enter a block number and then BLOCK followed by this ideogram, ensuring that this block will be written back to disk when the buffer's space is needed. The buffer address from BLOCK remains on the stack unchanged.

Comment: This ideogram is particularly useful when blocks are accessed for the addition or modification of data rather than as source screens.

USE

--- addr

MVP-FORTH

A variable containing the address of the block buffer to use next, as the least recently written. (*Fig*) Same as MVP. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
VARIABLE USE FIRST USE !
```

Fig: 8088/86:

```
BUF1 VARIABLE USE
```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

```
: <R/W>
  USE @ >R ROT USE ! SWAP MAX-DRV 0
  DO I DR-DEN DENSITY !
    DUP BPDRV - -1
    IF BPDRV - I 1+ MAX-DRV =
      IF R> R> DDROP R> USE !
      1 ABORT" BLOCK OUT OF RANGE" THEN
    ELSE I DRIVE ! LEAVE
  THEN
  LOOP SPBLK * SPBLK 0
  DO DDUP T&SCALC
  IF SEC-READ
```

```

ELSE SEC-WRITE
THEN 1+ HDBT 4 - SPBLK / USE +!
LOOP
DDROP R> USE ! ;

```

This example comes from the MVP-FORTH implementation.

Comment: This ideogram is important to the implementation dependent block I-O and buffer management routines, but it is of little value to applications.

USER

n ---

MVP-FORTH

A defining word which creates a user variable <name>. n is the cell offset within the user area where the value for <name> is stored. Execution of <name> leaves its absolute user area storage address. (Fig) A defining word used in the form: n USER cccc, which creates a user variable cccc. The parameter field of cccc contains n as a fixed offset relative to the user pointer register UP for this user variable. When cccc is later executed, it places the sum of its offset and the user area base address on the stack as the storage address of that particular variable. (79S *Reserved Word Set*) Same as MVP. (83S) Not defined. (F83) Vocabulary that holds task versions of defining words. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

```

: USER CONSTANT ;CODE
  DX INC  DX BX, MOV  [BX] BL, MOV  BH BH, SUB
  UP DI, MOV  [BX+DI] AX, LEA  APUSH JMP  END-CODE

```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

VOCABULARY USER

F-PC:

VOCABULARY USER

Example:

```

08 USER SP0

```

Enter the offset from the beginning of the user field, followed by the ideogram and then the name being assigned to that location.

Comment: A number of variables present in 79S could be defined with

VARIABLE

this construction, though the ideogram is not a part of the 79S vocabulary. It is, however, in the *Reference Word Set* and is customarily utilized, headerless if necessary. A group of them is usually initialized with data from low memory when Forth is started. In MVP-FORTH and in most implementations of fig-Forth, a block of 64 bytes is reserved for the user area. The use of cells in the 79S *Reserved Word Set* definition is interpreted to mean two bytes but not that word boundaries are required. The unused portion of this area is available to the programmer.

VALUE

NOT USED IN MVP-FORTH

n | <name> ---

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) A defining word that creates values. At run time the contents of the value is placed on the stack. The contents of the value may be changed by using =: .

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

: VALUE HEADER , JUMP , ;USES DOVALUE ,--X

Example:

```
10 VALUE TEN
TEN .
```

The value of TEN is defined as the integer 10. TEN then returns the integer 10 which is printed.

Comment: In F-PC the value can be changed with the ideogram =: . This is one implementation based on the work of Paul Bartholdi who used TO in place of =: to change the value.

VARIABLE

MVP-FORTH

A defining word to create a dictionary entry for <name> and allot two bytes for storage in the parameter field. The application must initialize

the stored value. When <name> is later executed, it will place the storage address on the stack. (*Fig*) A defining word used in the form: `n VARIABLE cccc`. When `VARIABLE` is executed, it creates the definition `cccc` with its parameter field initialized to `n`. When `cccc` is later executed, the address of its parameter field (containing `n`) is left on the stack, so that a fetch or store may access this location. (*79S*) Same as `MVP`. (*83S*) A defining word executed in the form: `VARIABLE <name>`. A dictionary entry for <name> is created and two bytes are `ALLOT`ed in its parameter field. This parameter field is to be used for contents of the variable. The application is responsible for initializing the contents of the variable which it creates. When <name> is later executed, the address of its parameter field is placed on the stack. (*F83*) A defining word to create variables. At runtime the address of the variable is placed on the stack. (*F-PC*) Same as `F83`.

Form: `VARIABLE <name>`

Implementations:

`MVP: 8088/86:`

```
: VARIABLE
  CREATE 2 ALLOT ;
```

`Fig: 8088/86:`

```
: VARIABLE
  CONSTANT ;CODE
  HERE LABEL DOVAR DX INC DX PUSH NEXT JMP
```

`F83: 8088/86:`

```
: VARIABLE
  CREATE 0 , ;USES DOCREATE ,
```

`F-PC:`

```
: VARIABLE
  CREATE 0 , ;USES >NEXT , -X
```

Example:

```
VARIABLE NEW-VALUE 0 NEW-VALUE !
```

This ideogram will define the name of a new variable which is then initialized.

Comment: There is a significant difference between the current *79S* function of this ideogram and that used by *fig-Forth*. The *MVP-FORTH* implementation conforms with *79S*. The current function of this ideogram is to create a dictionary space for a variable and leave its contents undetermined. Thus all variables must be initialized after being de-

VIEW

defined. This differs from fig-Forth in which the defined function of **VARIABLE** used the top value on the stack to initialize the variable at creation time.

VIEW | name --- NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Not defined. (F-PC) Implemented but not defined.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: VIEW [ DOS ] ' @VIEW ?DUP
  IF 2* VIEW-FILES + @ DUP
    IF ." is in " 2DUP >BODY .FILE
      " screen " EXECUTE
    ELSE 2DROP ." no entry " THEN
  ELSE ." may be in current file: " FILE?
    ." screen " DUP .
  THEN LIST ;
```

F-PC:

```
: VIEW
  >ON NEWBROWSE >ON ?BROWSE
  >OFF SEDING >OFF NEWFL
  >IN @ BL WORD SWAP >IN ! C@
  IF BL WORD HFIND 0= ?MISSING CFA_VIEW
  ELSE <ED>
  THEN ;
```

Example:

VIEW VIEW

The example will display its implementation in F-PC.

Comment: This ideogram has been implemented in a number of Forth dialects and is very convenient for exploring one you are not familiar with.

VLIST

MVP-FORTH UTILITY

List the word names of the CONTEXT vocabulary starting with the most recent definition. (*Fig*) List the names of the definitions in the context vocabulary. "Break" will terminate the listing. (*79S Reference Word Set*) Same as MVP. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

HEX

: VLIST

```

C/L OUT ! CONTEXT @ @
BEGIN C/L OUT @ - OVER C@
IF AND 4 + <
IF CR 0 OUT ! THEN
DUP ID. SPACE SPACE PFA 4 - @ DUP
NOT PAUSE ?TERMINAL OR
UNTIL DROP ;

```

Fig: 8088/86:

HEX

: VLIST

```

80 OUT ! CONTEXT @ @
BEGIN OUT @ C/L >
IF CR 0 OUT ! ENDIF
DUP ID. SPACE SPACE PFA LFA @ DUP 0=
?TERMINAL OR
UNTIL DROP ;

```

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

VLIST

This ideogram will start a printing of the CONTEXT vocabulary.

Comment: In most fig-Forth based implementations, pressing any key will terminate the listing. The MVP-FORTH version incorporates a PAUSE feature, by which pressing any key will freeze the display. Once suspended, the VLIST may be resumed by a single keystroke, or aborted by striking any two keys in rapid succession.

VOCABULARY

VOC-LINK

--- addr

MVP-FORTH

A user variable containing the address of a field in the definition of the most recently created vocabulary. All vocabulary names are linked by these fields to allow control for FORGETting through multiple vocabularies. (*Fig*) Same as MVP. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Points to the most recently defined vocabulary. (*F-PC*) Same as F83.

Implementations:

MVP: 8088/86:

HEX 14 USER VOC-LINK

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

VARIABLE VOC-LINK

F-PC:

Same as F83.

Example:

VOC-LINK @ U.

Get the address of the user variable, fetch its value and print it.

Comment: An implementation dependent variable not available in all versions of Forth.

VOCABULARY

MVP-FORTH

A defining word to create (in the CURRENT vocabulary) a dictionary entry for <name>, which specifies a new ordered list of word definitions. Subsequent execution of <name> will make it the CONTEXT vocabulary. When <name> becomes the CURRENT vocabulary (see DEFINITIONS), new definitions will be created in that list. In lieu of any further specifications, new vocabularies chain to FORTH. That is, when a dictionary search through a vocabulary is exhausted, FORTH will be searched. (*Fig*) A defining word used in the form: VOCABULARY cccc, to create a vocabulary definition cccc. Subsequent use of cccc will make it the CONTEXT vocabulary which is searched first by INTERPRET. The sequence "cccc DEFINITIONS" will also make cccc the CURRENT vocabulary into which new definitions are placed. In fig-Forth, cccc will be so chained as to include all definitions of the vocabulary in which cccc is itself defined. All vocabularies ultimately chain to Forth.

By convention, vocabulary names are to be declared IMMEDIATE. See VOC-LINK (79S) Same as MVP. (83S) A defining word executed in the form: VOCABULARY <name>. A dictionary entry for <name> is created which specifies a new ordered list of word definitions. Subsequent execution of <name> replaces the first vocabulary in the search order with <name>. When <name> becomes the compilation vocabulary new definitions will be appended to <name>'s list. See: DEFINITIONS "search order" (F83) Defines a new Forth vocabulary. VOC-LINK is a chain in temporal order and used by FORGET. At runtime a vocabulary changes the search order by setting CONTEXT. (F-PC) Defines a new Forth vocabulary.

Form: VOCABULARY <name>

Implementations:

MVP: 8088/86:

```
: VOCABULARY    'VOCABULARY @ EXECUTE ;
```

Fig: 8088/86:

```
HEX
: VOCABULARY
  <BUILDS 0A01 , CURRENT @ CFA ,
  HERE VOC-LINK @ , VOC-LINK !
  DOES 2+ CONTEXT ! ;
```

F83: 8088/86:

```
: VOCABULARY
  CREATE #THREADS 0 DO 0 , LOOP
  HERE VOC-LINK @ , VOC-LINK !
  DOES> CONTEXT ! ;
```

F-PC:

```
: VOCABULARY
  CREATE #THREADS 0
  DO 0 , LOOP
  HERE VOC-LINK @ , VOC-LINK !
  DOES> CONTEXT ! ;
```

Example:

```
VOCABULARY EDITOR
```

Create a new vocabulary name in Forth named the EDITOR.

Comment: Though in fig-Forth daughter vocabularies may be chained to parent vocabularies before finally chaining to Forth, the common interpretation of 79S is that all vocabularies must chain only to Forth.

WARM

Although there are some applications in which multiple daughter vocabularies might be desirable, in general the consensus of many Forth programmers is that the number of vocabularies should be kept a minimum.

For convenience VOCABULARY is vectored. The strict 79S implementation has the code field address of the run-time routine VOCABULARY79 in the variable 'VOCABULARY. Those desiring a different functional definition may write their own and place that code field address in 'VOCABULARY. The function in fig-Forth may be easily substituted by placing the code field address of VOCABULARYFIG which is implemented in MVP-FORTH, into 'VOCABULARY.

WARM

NOT USED IN MVP-FORTH

(Fig) 8080 - Restart Forth with EMPTY-BUFFERS . (Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Perform a warm start, jumped to by vector at hex 104. (F-PC) The WARM entry point for Forth, just calls the DEFERred word WARMFUNC. A WARM start is invoked whenever the "Control Break" key is pressed.

Implementations:

MVP: 8088/86: Could be defined as:

```
: WARM
  EMPTY-BUFFERS  ABORT  ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
: WARM
  TRUE ABORT" Warm Start"  ;
```

F-PC:

```
: WARM
  [ LABEL WARMBODY ]  WARMFUNC  ;
```

Example:

WARM

This ideogram will simply clear the buffers and restart Forth.

Comment: As implemented in the 8080 version of fig-Forth, this ideogram simply empties the buffers and clears the stack. Nothing in the dictionary is changed.

WARNING

--- addr

MVP-FORTH

A user variable containing a flag which enables the output of selected non-fatal error messages. (*Fig*) A user variable containing a value controlling messages. If = 1 disc is present, and screen 4 of drive 0 is the base location for messages. If = 0, no disc is present and messages will be presented by number. If = -1 execute (ABORT) for a user specified procedure. See MESSAGE, ERROR. (79S) Not defined. (83S) Not defined. (F83) Checked by WARN for duplicate warnings. (F-PC) Checked by WARN for duplicate warnings.

Implementations:

MVP: 8088/86:

HEX 0E USER WARNING

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

VARIABLE WARNING

F-PC:

Same as F83.

Example:

0 WARNING 1

Suppress the printing of system warning messages.

Comment: Within the MVP-FORTH nucleus, this flag controls the "ISN'T UNIQUE" message. It is available for the programmer's use. Note that fig-Forth uses this user variable in a completely different manner: to control the source of all message texts.

WHERE

MVP-FORTH

Display the last character string parsed by the text interpreter, along with the line containing it. If loading, the screen and line numbers are printed. (*Fig*) Not defined. (79S) Not defined. (83S) Not defined. (F83) Locates the screen and position following an error. (F-PC) Not defined.

Implementations:

MVP: 8088/86:

: WHERE

BLK @

WHILE

```
IF BLK @ DUP SCR ! CR CR ." SCR# "  
  DUP . >IN @ 3FF MIN C/L /MOD DUP  
  ." LINE# " . C/L * ROT BLOCK +  
  CR CR C/L -TRAILING TYPE  
  >IN @ 3FF > +  
ELSE >IN @  
THEN CR HERE C@ DUP >R - HERE R@ +  
1+ C@ 20 =  
IF 1- THEN SPACES R> 0  
DO 5E EMIT LOOP ;
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
DEFER WHERE  
: (WHERE)  
  DISK-ERROR @ 0= FILE @ 0<> AND  
  IF EDIT [ EDITOR ] 1- C 'WORD COUNT 'FIND PLACE  
  ELSE DONE THEN ;  
  ' (WHERE) IS WHERE
```

F-PC:

Not used.

Example:

```
: ABORT"  
  IF WHERE CR R@ COUNT TYPE SP! QUIT  
  ELSE >R DUP C@ + 1+ R> THEN ;
```

This example from the MVP-FORTH source code shows how ABORT" tells you exactly where the text interpreter encountered an error.

Comment: WHERE makes use of the count and character string stored at HERE, to identify the last ideogram interpreted. It is useful for locating the source of common errors during LOADING or terminal interpretation, but is no substitute for good debugging skills.

WHILE

flag ---

MVP-FORTH

Used in a colon-definition to select conditional execution based on the flag. On a true flag, continue execution through to REPEAT, which then returns back to just after BEGIN. On a false flag, skip execution to just after REPEAT, exiting the structure. (*Fig*) Occurs in a colon-definition in the form: BEGIN ... WHILE (tp) ... REPEAT. At run-time, WHILE selects conditional execution based on boolean flag f. If f is true

(non-zero), WHILE continues execution of the true part through to REPEAT, which then branches to BEGIN. If f is false (zero) execution skips to just after REPEAT, exiting the structure. At compile time, WHILE emplaces (OBRANCH) and leaves ad2 of the reserved offset. The stack values will be resolved by REPEAT. (79S) Same as MVP. (83S) Selects conditional execution based on flag. When flag is true, execution continues to just after the WHILE through to the REPEAT which then continues execution back to just after the BEGIN. When flag is false, execution continues to just after the REPEAT, exiting the control structure. sys1 is balanced with its corresponding BEGIN. sys2 is balanced with its corresponding REPEAT. See: BEGIN (F83) Not defined. (F-PC) Used in the form: BEGIN <loop> flag WHILE <true> REPEAT. Repeat <loop> and <true> clauses while the flag f1 is true (really, non-zero).

Form: BEGIN ... flag WHILE ... REPEAT

Implementations:

MVP: 8088/86:

```
: WHILE
  [COMPILE] IF 2+ ; IMMEDIATE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
: WHILE
  COMPILE ?BRANCH >MARK 2000 ; IMMEDIATE
```

F-PC:

```
: WHILE
  COMPILE ?WHILE ?>MARK 2SWAP ; IMMEDIATE
```

Example:

```
: TEST 0
  BEGIN DUP 10
  WHILE DUP . 1+
  REPEAT DROP ;
```

The definition illustrates this ideogram in another BEGIN construct which will type the digits 0 through 9.

Comment: WHILE makes its exit decision in the middle of the loop. Thus the test can be made and the loop exited before passing through the body of the loop at all. Compare this with the DO ... LOOP and BEGIN ... UNTIL loops which always make at least one complete pass

WITHIN

through the body of the loop before testing the exit criterion.

WIDTH

--- addr

MVP-FORTH

In fig-Forth, a user variable containing the maximum number of letters saved in the compilation of a definition's name. It must be 1 through 31, with a default value of 31. The name character count and its natural characters are saved, up to the value in WIDTH. The value may be changed at any time within the above limits. (*Fig*) Same as MVP. (79S) Not defined. (83S) Not defined. (F83) Number of characters to keep in name field. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

HEX 0C USER WIDTH

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

VARIABLE WIDTH

F-PC:

Same as F83.

Example:

WIDTH @ .

Get the address of this user variable, then fetch its value and print it to show the present maximum size of a name in the dictionary header.

Comment: Though ideograms may have up to 31 characters in 79S, the MVP- FORTH implementation gives you the option of truncating them to the length specified in this user variable.

WITHIN

NOT USED IN MVP-FORTH

n min max -- f

(*Fig*) Not defined. (79S) Not defined. (83S) Not defined (F83) Return true if $\text{min} < n1 < \text{max}$, otherwise false. (F-PC) Return true if $l0 \leq n < h1$, otherwise false. Signed comparison.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

: WITHIN 1- BETWEEN ;

F-PC:

CODE WITHIN

```
POP DI    POP CX    POP DX
XOR AX, AX  CMP DX, DI  <
IF    CMP DX, CX  >=
    IF    DEC AX    THEN
    THEN    1 PUSH    END-CODE
```

Example:

12 1 12 WITHIN

The value 12 is not WITHIN the upper limit of 12 and a false flag is returned.

Comment: Note the similarity with BETWEEN where the test is with the upper limit inclusive. Both WITHIN and BETWEEN return a true flag when the number is equal to the lower limit.

WORD

char --- addr

MVP-FORTH

Receive characters from the input stream until the non-zero delimiting character is encountered or the input stream is exhausted, ignoring leading delimiters. The characters are stored as a packed string with the character count in the first character position. The actual delimiter encountered (char or null) is stored at the end of the text but not included in the count. If the input stream was exhausted as WORD is called, then a zero length will result. The address of the beginning of this packed string is left on the stack. (*Fig*) Read the next text characters from the input stream being interpreted, until a delimiter *c* is found, storing the packed character string beginning at the dictionary buffer HERE. WORD leaves the character count in the first byte, the characters, and ends with two or more blanks. Leading occurrences of *c* are ignored. If BLK is zero, text is taken from the terminal input buffer, otherwise from the disc block stored in BLK. See BLK, IN. (79S) Same as MVP. (83S) Generates a counted string by non-destructively accepting characters from the input stream until the delimiting character *char* is encountered or the input stream is exhausted. Leading delimiters are ignored. The entire character string is stored in memory beginning at *addr* as a sequence of bytes. The string is followed by a blank which is not included in the count. The first byte of the string is

WORD

the number of characters {0..255}. If the string is longer than 255 characters, the count is unspecified. If the input stream is already exhausted as WORD is called, then a zero length character string will result. If the delimiter is not found the value of >IN is the size of the input stream. If the delimiter is found >IN is adjusted to indicate the offset to the character following the delimiter. #TIB is unmodified. The counted string returned by WORD may reside in the "free" dictionary area at HERE or above. Note that the text interpreter may also use this area. See: "input stream" (F83) Parse the input stream for char and return a count delimited string at here. Note there is always a blank following it. (F-PC) Same as F83.

Implementations:

MVP: 8088/86:

```
: WORD
  'WORD @ EXECUTE ;
```

Fig: 8088/86:

```
HEX
: WORD
  BLK @
  IF BLK @ BLOCK ELSE TIB @ ENDIF
  IN @ + SWAP ENCLOSE HERE 22 BLANK
  IN +! OVER - >R R HERE C! + HER 1+ R> CMOVE ;
```

F83: 8088/86:

```
: 'WORD HERE ;
: WORD
  PARSE-WORD 'WORD PLACE
  'WORD DUP COUNT + BL SWAP C!
  ( Stick Blank at end ) ;
```

F-PC:

```
CODE WORD
  MOV DI, 'TIB  MOV CX, #TIB  POP BX  PUSH ES
  MOV DX, DS  MOV ES, DX
  MOV AX, >IN  CMP CX, AX  U<=
  IF  MOV AX, CX  THEN
  ADD DI, AX  SUB CX, AX  MOV AX, BX  CX<>0
  IF  REPZ  SCASB  0<>
    IF  INC CX  DEC DI  THEN
  THEN  MOV DX, DI  MOV AX, BX  CX<>0
  IF  REPNZ  SCASB  0=
    IF  INC CX  DEC DI  THEN
```



```

    THEN    SUB DI, DX    MOV BX, #TIB    MOV AX, DX
CX<>0
    IF      DEC CX      THEN
    SUB BX, CX    MOV >IN BX    MOV BX, UP
    MOV DX, DP [BX]    MOV CX, DI    MOV DI, DX
    MOV 0 [DI], CL    INC DI    MOV BX, IP
    MOV IP, AX    REPNZ    MOVSB    MOV AL, # 32
    STOSB    MOV IP, BX    POP ES    PUSH DX
    NEXT      END-CODE

```

Example:

```

: INPUT  ."  Input an integer  --> "
  QUERY  BL  WORD  NUMBER  DROP  ;

```

The definition provides for a prompt and then a pause for the operator to input the requested integer. Then the input character stream is parsed with this ideogram, converted to a double precision value and reduced to a single precision value left on the stack.

Comment: Care must be taken in moving source code from fig-Forth to 79S which includes the ideogram WORD. In 79S, WORD leaves the address of HERE on the top of the stack. Also, while in fig-Forth the string is stored at HERE, in 79S another buffer may be used. Because of the possible variations, this ideogram utilizes an execution vector which will permit easy redefinition. It defaults to WORD.

WORDS | text text --- NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) List the words in the context vocabulary. This can be interrupted any time by pressing any key. (F-PC) Display words that match text. <text> is optional, if omitted then the CONTEXT vocabulary will be displayed. Two space delimited <text> strings may follow, and only words containing both <text> strings will be printed:

```

WORDS  >> @  <enter>
          display all words containing both @  and .

WORDS  X Y  <enter>
          display all words containing both X  and Y.

WORDS  DUP  <enter>
          display all words containing DUP.

```

Implementations:

MVP: 8088/86:

X

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: WORDS
  NEWLINE [ ' ] .NAME OVERVOC ;
```

F-PC:

```
: WORDS
TOTALWORDS OFF SAVESTATE COLS 2- RMARGIN !
15 TABSIZE ! LMARGIN OFF
CR ." ** Press SPACE to pause, or ESC to exit ** "
PREWORDS >IN @ #TIB @ <>
IF [ ' ] ?INNAME IS ?W.NAME
  BL WORD W$ OVER C@ 1+ 32 MIN CMOVE
  BL WORD W$ 32 + OVER C@ 1+ 32 MIN CMOVE
  ?*.* ?CODE.* ?:.? ?VARIABLE.* ?CONSTANT.*
  ?DEFER.* ?VALUE.* ?UVARIABLE.* ?UDEFER.*
  ?TOTAL.* CONTEXTONLY
  FALSE =: CONTEXTONLY
  IF CONTEXT @ .VOCWORDS
  ELSE VOC-LINK @
    BEGIN DUP #THREADS 2* - .VOCWORDS @ DUP 0=
    UNTIL DROP
  THEN
  ELSE [ ' ] NOOP IS ?W.NAME
    FALSE =: CONTEXTONLY
    CONTEXT @ .VOCWORDS
  THEN CR TOTALWORDS @ U. ." Words displayed" CR
  RESTORESTATE ;
```

Example:

```
WORDS XYZ
```

The example will display all words in the CONTEXT vocabulary containing XYZ.

Comment: This ideogram has been implemented in a number of dialects of Forth. It is particularly convenient when you are new to a dialect.

X

MVP-FORTH

This is a pseudonym for the "null" or dictionary entry for a name of one character of ASCII null. It is the execution procedure to terminate

interpretation of a line of text from the terminal or within a disk buffer, as both buffers always have a null at the end. (*Fig*) Same as MVP. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) Not defined.

Implementations:

MVP: 8088/86:

```
: X
  BLK @
  IF STATE @ ?STREAM THEN
  R> DROP ; IMMEDIATE IS-X
```

Fig: 8088/86:

Not used.

F83: 8088/86:

Not used.

F-PC:

Not used.

Example:

This ideogram is not available to the programmer.

Comment: This somewhat confusing word is defined in the *Installation Manual* but is not included in *79S*. Defining the null character as an ideogram is a clever trick for telling <INTERPRET> that the end of a buffer has been reached. Note the intentionally unbalanced use of R. This ideogram IS-X, is part of the CROSS-COMPILER. It locates the "X" in the latest definition's dictionary and overwrites it with a null. The sequence "0 ' X NFA 1+ C!" would accomplish the same thing.

XOR

n1 n2 --- n3

MVP-FORTH

Leave the bitwise exclusive-or of two numbers. (*Fig*) Leave the bitwise logical exclusive-or of two values. (*79S*) Same as MVP. (*83S*) 16b3 is the bit-by-bit exclusive-or of 16b1 with 16b2. (*F83*) Returns the bitwise exclusive or of n1 and n2 on the stack. (*F-PC*) Same as F83.

Pronounced: x-or

Implementations:

MVP: 8088/86:

```
CODE XOR
  AX POP    BX POP    BX AX, XOR
  APUSH JMP  END-CODE
```

[

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

CODE XOR

BX POP AX POP BX AX XOR 1PUSH

F-PC:

Same as F83.

Example:

HEX 20 10 XOR DECIMAL

The example does a logical exclusive-or operation on the bit patterns which are perhaps more easily recognized when in hex. The two values are replaced by the result on the stack.

Comment: This ideogram is a common logical operator.

[

MVP-FORTH

End the compilation mode. The text from the input stream is subsequently executed. See:] (*Fig*) Used in a colon-definition in form : xxx [words] more ; Suspend compilation. The words after [are executed, not compiled. This allows calculation or compilation exceptions before resuming compilation with]. See LITERAL (79S) Same as MVP. (83S) Sets interpret state. The text from the input stream is subsequently interpreted. For typical usage see LITERAL . See:] (*F83*) Stop compiling and start interpreting (*F-PC*) Same as F83.

Pronounced: left-bracket

Implementations:

MVP: 8088/86:

```
: [
  0 STATE ! ; IMMEDIATE
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
: [
  STATE OFF [ ' ] INTERPRETER IS EVAL ; IMMEDIATE
```

F-PC:

```
: [
  STATE OFF ; IMMEDIATE
```

Example:

```
: ADD-RECORDLENGTH [ BLOCKSIZE
  RECORDS-PER-BLOCK / ] LITERAL + ;
```

This ideogram [suspends compilation so that the division of two constants is done, only once, at compile time. Of course, the constants will have been defined earlier.

Comment: Allows switching out of the compile mode within a definition. It can be used for compile-time calculation, SMUDGE-ing the name of a recursive definition, or anything else. Take care not to interfere with the compiler's use of the stack, as inside IF or BEGIN constructs.

['] --- MVP-FORTH SUPPLEMENTAL

Used in a colon definition to compile the parameter field address of the next word in the input stream as a literal. (*Fig*) Not defined. (79S) Not defined. (83S) Used in the form: ['] <name> . Compiles the compilation address *addr* of <name> as a literal. When the colon definition is later executed *addr* is left on the stack. An error condition exists if <name> is not found in the currently active search order. See. LITERAL (F83) Compile the code field of the next word as a literal. (F-PC) Like ' only used while compiling.

Form:

```
--- (compile time)
--- adr (run time)
```

Pronounced: bracket-tick-bracket

Implementations:

MVP: 8088/86:

```
: [ ' ]
  ?COMP [COMPILE] ' ; IMMEDIATE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: [ ' ]
  ' COMPILE ( ' ) , ; IMMEDIATE
```

F-PC:

```
: [ ' ]
  ' COMPILE <'> X, ; IMMEDIATE
```

[COMPILE]

Example:

```
: TEST    [ ' ]  B/BUF  ;
```

The parameter field address of B/BUF is compiled as a literal into the definition of TEST. TEST will behave like a constant, always pushing the same literal value onto the stack.

Comment: This form of ' is used in colon definitions apparently to keep things straight in poly-FORTH. Note that the address returned by ['] should not be EXECUTED in this version until it has been adjusted by the operator CFA.

[COMPILE]

MVP-FORTH

Used in a colon-definition to force compilation of the following word. This allows compilation of an IMMEDIATE word when it would otherwise be executed. (*Fig*) Used in a colon-definition in form: : xxx [COMPILE] FORTH ;. [COMPILE] will force the compilation of an immediate definition, that would otherwise execute during compilation. The above example will select the FORTH vocabulary when xxx executes, rather than at compile time. (79S) Same as MVP. (83S) Used in the form: [COMPILE] <name>. Forces compilation of the following word <name>. This allows compilation of an immediate word when it would otherwise have been executed. (*F83*) Force compilation of an immediate word (*F-PC*) - Same as F83.

Form: [COMPILE] <name>

Pronounced: bracket-compile

Implementations:

MVP: 8088/86:

```
: [COMPILE]
  ?COMP -FIND NOT
  ABORT" NOT FOUND" DROP CFA , ;
IMMEDIATE
```

Fig: 8088/86:

```
: [COMPILE]
  -FIND 0= 0 ?ERROR DROP CFA , ;
```

F83: 8088/86:

```
: [COMPILE]
  ' , ; IMMEDIATE
```

F-PC:

```
: [COMPILE]
  ' X,    ; IMMEDIATE
```

Example:

```
: ENDIF    [COMPILE] THEN ; IMMEDIATE
```

The example illustrates the renaming of an ideogram which is marked immediate.

Comment: This is another ideogram which gives versatility to the defining ideograms in Forth. It overrides an ideogram's precedence bit. The function of this ideogram is tricky especially when dealing with state smart words.

\ --- MVP-FORTH UTILITY

Ignore all subsequent characters on this line. Begin interpretation with the next line. May be used only while loading. (*Fig*) Not defined. (*79S*) Not defined. (*83S*) Not defined. (*F83*) Not defined. (*F-PC*) - This line is a comment till the end of this line.

Pronounced: back-slash

Implementations:

MVP: 8088/86:

```
: \
  >IN @ C/L / 1+
  C/L * >IN ! ;
IMMEDIATE
```

Fig: 8088/86:

Not used.

F83: 8088/86:

```
: \
  >IN @ NEGATE C/L MOD >IN +! ; IMMEDIATE
```

F-PC:

```
: \
  #TIB @ >IN ! ; IMMEDIATE
```

Example:

```
/    THIS IS A COMMENT
```

The interpreter will ignore this text.

Comment: This ideogram offers an alternative method for commenting a source program. While it has the advantage of not requiring a closing

1

delimiter, it is non-standard and works only for the 16 x 64 screen format.

\S **---**

NOT USED IN MVP-FORTH

(Fig) Not defined. (79S) Not defined. (83S) Not defined. (F83) Comment to end of screen. (F-PC) Stop loading the current file with the line that contains this word.

Implementations:

MVP: 8088/86:

Not used.

Fig: 8088/86:

Not used.

F83: 8088/86:

: \S

SOURCE NIP >IN ! ; IMMEDIATE

F-PC:

: /S

```
SEOHANDLE  ENDFILE  2DROP  LOADLINE  OFF
```

```
IBRESET    %@    #TIB %!    >IN    ;
```

Example:

```
\S The rest of this screen is a comment.
```

In the example, compilation of the screen will stop at the ideogram in F83. The example in F-PC is different: the rest of the input file is ignored.

Comment: Different implementations of Forth will have different functions and implementations for this ideogram.

1. ---

MVP-FORTH

Set the compilation mode. The text from the input stream is subsequently compiled. See [. (*Fig*) Resume compilation, to the completion of a colon-definition. See [. (79S) Same as MVP. (83S) Sets compilation state. The text from the input stream is subsequently compiled. For typical usage see LITERAL. See: [(*F83*) The Compiling Loop. First sets Compile State. Looks up the next word in the input stream and either executes it or compiles it depending upon whether or not it is immediate. If the word is not in the dictionary, it converts it to a number, either single or double precision depending on whether

or not any punctuation was present. Continues until input stream is empty or state changes. (*F-PC*) Same as F83.

Pronounced: right-bracket

Implementations:

MVP: 8088/86:

```
: ]
    C0 STATE ! ;
```

Fig: 8088/86:

Same as MVP.

F83: 8088/86:

```
: ]
    STATE ON [ ' ] COMPILER IS EVAL ;
```

F-PC:

```
DEFER ]
: ( )
    STATE ON
    BEGIN ?STACK DEFINED DUP
    IF 0>
        IF EXECUTE ELSE X, THEN
        ELSE DROP NUMBER DOUBLE?
        IF [COMPILE] DLITERAL
        ELSE DROP [COMPILE] LITERAL
        THEN THEN TRUE DONE?
    UNTIL ;
DEFER ] ' ( ) IS ]
```

Example:

```
: ADD-RECORDLENGTH [ BLOCKSIZE
    RECORDS-PER-BLOCK / ] LITERAL + ;
```

This ideogram,] , resumes compilation after a compile-time calculation has been performed.

Comment: This ideogram works with [to allow access to Forth's execution-time functions within a colon definition. Take care not to disturb stack data being used by the compiler, as in IF, BEGIN and DO-LOOP conditional constructs.

Different systems are known to handle this function in different ways. It is a place to look for possible trouble.

**fig-FORTH
INSTALLATION MANUAL**

**GLOSSARY
MODEL
EDITOR**

RELEASE 1
WITH COMPILER SECURITY
AND
VARIABLE LENGTH NAMES

BY
WILLIAM F. RAGSDALE

November 1980

Provided through the courtesy of the FORTH INTEREST GROUP, PO Box 1105,
San Carlos, CA 94070

Further distribution of this public domain publication must include this
notice.

FORTH INTEREST GROUP.....PO BOX 1105.....San Carlos, CA 94070

fig-FORTH INSTALLATION MANUAL

1.0	INTRODUCTION
2.0	DISTRIBUTION
3.0	MODEL ORGANIZATION
4.0	INSTALLATION
5.0	MEMORY MAP
6.0	DOCUMENTATION SUMMARY

1.0 INTRODUCTION

The fig-FORTH implementation project occurred because a key group of Forth fanciers wished to make this valuable tool available on a personal computing level. In June of 1978, we gathered a team of nine systems level programmers, each with a particular target computer. The charter of the group was to translate a common model of Forth into assembly language listings for each computer. It was agreed that the group's work would be distributed in the public domain by FIG. This publication series is the conclusion of the work.

2.0 DISTRIBUTION

All publications of the Forth Interest Group are public domain. They may be further reproduced and distributed by inclusion of this credit notice:

This publication has been made available by the Forth Interest Group,
P. O. Box 1105, San Carlos, Ca 94070

We intend that our primary recipients of the Implementation Project be computer users groups, libraries, and commercial vendors. We expect that each will further customize for particular computers and redistribute. No restrictions are placed on cost, but we

expect faithfulness to the model. FIG does not intend to distribute machine readable versions, as that entails customization, revision, and customer support better reserved for commercial vendors.

Of course, another broad group of recipients of the work is the community of personal computer users. We hope that our publications will aid in the use of Forth and increase the user expectation of the performance of high level computer languages.

3.0 MODEL ORGANIZATION

The fig-FORTH model deviates a bit from the usual loading method of Forth. Existing systems load about 2k bytes in object form and then self-compile the resident system (6 to 8 k bytes). This technique allows customization within the high level portion, but is impractical for new implementors.

Our model has 4 to 5 k bytes written as assembler listings. The remainder may be compiled typing in the Forth high-level source, by more assembly source, or by disc compilation. This method enhances transportability, although the larger portion in assembly code entails more effort. About 8k bytes of memory is used plus 2 to 8k for workspace.

3.1 MODEL OVER-VIEW

The model consists of 7 distinct areas. They occur sequentially from low memory to high.

- Boot-up parameters
- Machine code definitions
- High level utility definitions
- Installation dependent code
- High level definitions
- System tools (optional)
- RAM memory workspace

FORTH INTEREST GROUP P.O. Box 1105 San Carlos, Ca. 94070

3.2 MODEL DETAILS

Boot-up Parameters

This area consists of 34 bytes containing a jump to the cold start, jump to the warm re-start and initial values for user variables and registers. These values are altered as you make permanent extensions to your installation.

Machine Code Definitions

This area consists of about 600 to 800 bytes of machine executable code in the form of Forth word definitions. Its purpose is to convert your computer into a standard Forth stack computer. Above this code, the balance of Forth contains a pseudo-code compiled of "execution-addresses" which are sequences of the machine address of the "code-fields" of other Forth definitions. All execution ultimately refers to the machine code definitions.

High-level Utility Definitions

These are colon-definitions, user variables, constants, and variables that allow you to control the "Forth stack computer". They comprise the bulk of the system, enabling you to execute and compile from the terminal. If disc storage (or a RAM simulation of disc) is available, you may also execute and compile from this facility. Changes in the high-level area are infrequent. They may be made thru the assembler source listings.

Installation Dependent Code

This area is the only portion that need change between different installations of the same computer cpu. There are four code fragments:

(KEY) Push the next ascii value (7 bits) from the terminal keystroke to the computation stack and execute NEXT. High 9 bits are zero. Do not echo this character, especially a control character.

(EMIT) Pop the computation stack (16 bit value). Display the low 7 bits on the terminal device, then execute NEXT. Control characters have their natural functions.

(?TERMINAL) For terminals with a break key, wait till released and push to the computation stack 0001 if it was found depressed; otherwise 0000. Execute NEXT. If no break key is available, sense any key depression as a break (sense but don't wait for a key). If both the above are unavailable, simply push 0000 and execute NEXT.

(CR) Execute a terminal carriage return and line feed. Execute NEXT.

When each of these words is executed, the interpreter vectors from the definition header to these code sequences. On specific implementations it may be necessary to preserve certain registers and observe operating system protocols. Understand the implementors methods in the listing before proceeding!

R/W This colon-definition is the standard linkage to your disc. It requests the read or write of a disc sector. It usually requires supporting code definitions. It may consist of self-contained code or call ROM monitor code. When R/W is assembled, its code field address is inserted once in BLOCK and once in BUFFER.

An alternate version of R/W is included that simulates disc storage in RAM. If you have over 16 k bytes this is practical for startup and limited operation with cassette.

High-level Definitions

The next section contains about 30 definitions involving user interaction: compiling aids, finding, forgetting, listing, and number formatting. These definitions are placed above the installation dependent code to facilitate modification. That is, once your full system is up, you may FORGET part of the high-level and re-compile altered definitions from disc.

System Tools

A text editor and machine code assembler are normally resident. We are including a sample editor, and hope to provide Forth assemblers. The editor is compiled from the terminal the first time, and then used to place the editor and assembler source code on disc.

It is essential that you regard the assembly listing as just a way to get Forth installed on your system. Additions and changes must be planned and tested at the usual Forth high level and then the assembly routines updated. Forth work planned and executed only at an assembly level tends to be non-portable, and confusing.

RAM Workspace

For a single user system, at least 2k bytes must be available above the compiled system (the dictionary). A 16k byte total system is most typical.

The RAM workspace contains the computation and return stacks, user area, terminal input buffer, disc buffer and compilation space for the dictionary.

FORTH INTEREST GROUP P.O. Box 1105 San Carlos, Ca. 94070

2

4.0 INSTALLATION

We see the following methods of getting a functioning fig-FORTH system:

1. Buy loadable object code from a vendor who has customized.
2. Obtain an assembly listing with the installation dependent code supplied by the vendor. Assemble and execute.
3. Edit the FIG assembly listing on your system, re-write the I-O routines, and assemble.
4. Load someone else's object code up to the installation dependent code. Hand assemble equivalents for your system and poke in with your monitor. Begin execution and type in (self-compile) the rest of the system. This takes

about two hours once you understand the structure of Forth (but that will take much more time!).

Let us examine Step 3, above, in fuller detail. If you wish to bring up Forth only from this model, here are the sequential steps:

- 4.1 Familiarize yourself with the model written in Forth, the glossary, and specific assembly listings.
- 4.2 Edit the assembly listings into your system. Set the boot-up parameters at origin offset 0A, 0B (bytes) to 0000 (warning=00).
- 4.3 Alter the terminal support code (KEY, EMIT, etc.) to match your system. Observe register protocol specific to your implementation!
- 4.4 Place a break to your monitor at the end of NEXT, just before indirectly jumping via register W to execution. W is the Forth name for the register holding a code field address, and may be differently referenced in your listings.
- 4.5 Enter the cold start at the origin. Upon the break, check that the interpretive pointer IP points within ABORT and W points to SP!. If COLD is a colon-definition, then the IP has been initialized on the way to NEXT and your testing will begin in COLD. The purpose of COLD is to initialize IP, SP, RP, UP, and some user variables from the start-up parameters at the origin.
- 4.6 Continue execution one word at a time. Clever individuals could write a simple trace routine to print IP, W, SP, RP and the top of the stacks. Run in this single step mode until the greeting message is printed. Note that the interpretation is several hundred cycles to this stage!

4.7 Execution errors may be localized by observing the above pointers when a crash occurs.

4.8 After the word QUIT is executed (incrementally), and you can input a "return" key and get OK printed, remove the break. You may have some remaining errors, but a reset and examination of the above registers will again localize problems.

4.9 When the system is interpreting from the keyboard, execute EMPTY-BUFFERS to clear the disc buffer area. You may test the disc access by typing: 0 BLOCK 64 TYPE This should bring sector zero from the disc to a buffer and type the first 64 characters. This sector usually contains ascii text of the disc directory. If BLOCK (and R/W) doesn't function--happy hunting!

5.0 If your disc driver differs from the assembly version, you must create your own R/W. This word does a range check (with error message), modulo math to derive sector, track, and drive and passes values to a sector-read and sector-write routine.

RAM DISC SIMULATION

If disc is not available, a simulation of BLOCK and BUFFER may be made in RAM. The following definitions setup high memory as mass storage. Referenced 'screens' are then brought to the 'disc buffer' area. This is a good method to test the start-up program even if disc may be available.

```

HEX
4000 CONSTANT LO ( START OF BUFFER AREA )
6800 CONSTANT HI ( 10 SCREEN EQUIVALENT )
: R/W >R ( save boolean )
  B/BUF * LO + DUP
  HI > 6 ?ERROR ( range check )
  R> IF ( read ) SWAP ENDIF
  B/BUF CMOVE ;

```

Insert the code field address of R/W into BLOCK and BUFFER and proceed as if testing disc. R/W simulates screens 0 thru 9 when B/BUF is 128, in the memory area \$4000 thru \$6BFF.

FORTH INTEREST GROUP P.O. Box 1105 San Carlos, Ca. 94070

fig-FORTH VARIABLE NAME FIELD

A major FIG innovation in this model, is the introduction of variable length definition names in compiled dictionary entries. Previous methods only saved three letters and the character count.

The user may select the letter count saved, up to the full natural length. See the glossary definition for WIDTH.

In this model, the following conventions have been established.

1. The first byte of the name field has the natural character count in the low 5 bits.
2. The sixth bit = 1 when smudged, and will prevent a match by (FIND).
3. The seventh bit = 1 for IMMEDIATE definitions; it is called the precedence bit.
4. The eighth or sign bit is always = 1.
5. The following bytes contain the names' letters, up to the value in WIDTH.
6. In the byte containing the last letter saved, the sign bit = 1.
7. In word addressing computer, a name may be padded with a blank to a word boundary.

The above methods are implemented in CREATE. Remember that -FIND uses BL WORD to bring the next text to HERE with the count preceding. All that is necessary, is to limit by WIDTH and toggle the proper delimiting bits.

5.0 MEMORY MAP

The following memory map is broadly used. Specific installations may require alterations but you may forfeit functions in future FIG offerings.

The disc buffer area is at the upper bound of RAM memory. It is comprised of an integral number of buffers, each B/BUF+4 bytes. B/BUF is the number of bytes read from the disc, usually one sector. B/BUF must be a power of two (64, 128, 256, 512 or 1024). The constant FIRST has the value of the address of the start of the first buffer. LIMIT has the value of the first address beyond the top buffer. The distance between FIRST and LIMIT must be N*(B/BUF+4) bytes. This N must be two or more.

Constant B/SCR has the value of the number of buffers per screen; i.e. 1024 / B/BUF.

The user area must be at least 34 bytes; 48 is more appropriate. In a multi-user system, each user has his own user area, for his copy of system variables. This method allows re-entrant use of the Forth vocabulary.

The terminal input buffer is decimal 80 bytes (the hex 50 in QUERY) plus 2 at the end. If a different value is desired, change the limit in QUERY. A parameter in the boot-up literals locates the address of this area for TIB. The backspace character is also in the boot-up origin parameters. It is universally expected that "rubout" is the backspace.

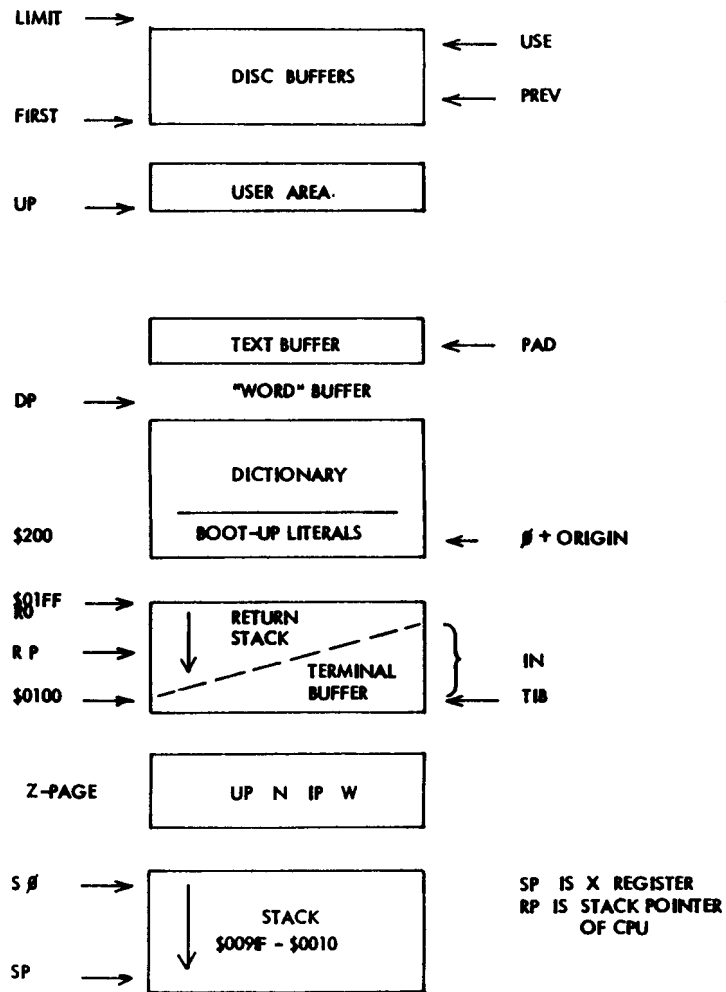
The return stack grows downward from the user area toward the terminal buffer. Forty-eight bytes are sufficient. The origin is in R0 (R-zero) and is loaded from a boot-up literal.

The computation stack grows downward from the terminal buffer toward the dictionary, which grows upward. The origin of the stack is in variable S0 (S-zero) and is loaded from a boot-up literal.

After a cold start, the user variables contain the addresses of the above memory assignments. An advanced user may relocate while the system is running. A newcomer should alter the startup literals and execute COLD. The word +ORIGIN is provided for this purpose. +ORIGIN gives the address byte or word relative to the origin depending on the computer addressing method. To change the backspace to control H type:

```
HEX 08 0E +ORIGIN ! ( byte addresses)
```


6502
fig-FORTH MEMORY MAP



STANDARD
fig-FORTH MEMORY MAP

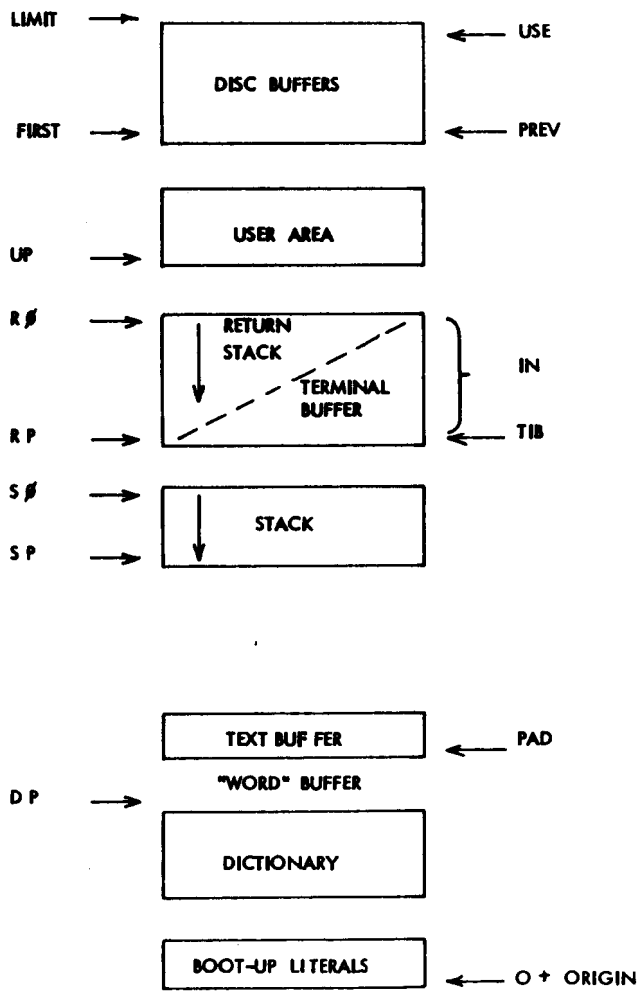


fig-FORTH GLOSSARY

This glossary contains all of the word definitions in Release 1 of fig-FORTH. The definitions are presented in the order of their ascii sort.

The first line of each entry shows a symbolic description of the action of the procedure on the parameter stack. The symbols indicate the order in which input parameters have been placed on the stack. Three dashes "---" indicate the execution point; any parameters left on the stack are listed. In this notation, the top of the stack is to the right.

The symbols include:

addr	memory address
b	8 bit byte (i.e. hi 8 bits zero)
c	7 bit ascii character (hi 9 bits zero)
d	32 bit signed double integer, most significant portion with sign on top of stack.
f	boolean flag. 0=false, non-zero=true
ff	boolean false flag=0
n	16 bit signed integer number
u	16 bit unsigned integer
tf	boolean true flag-non-zero

The capital letters on the right show definition characteristics:

C	May only be used within a colon definition. A digit indicates number of memory addresses used, if other than one.
E	Intended for execution only.
L0	Level Zero definition of FORTH-78
L1	Level One definition of FORTH-78
P	Has precedence bit set. Will execute even when compiling.
U	A user variable.

Unless otherwise noted, all references to numbers are for 16 bit signed integers. On 8 bit data bus computers, the high byte of a number is on top of the stack, with the sign in the leftmost bit. For 32 bit signed double numbers, the most significant part (with the sign) is on top.

All arithmetic is implicitly 16 bit signed integer math, with error and under-flow indication unspecified.

	n addr --- Store 16 bits of n at address. Pronounced "store".	L0	(+LOOP) The run-time procedure compiled by +LOOP, which increments the loop index by n and tests for loop completion. See +LOOP.	C2
ICSP	Save the stack position in CSP. Used as part of the compiler security.		(ABORT)	Executes after an error when WARNING is -1. This word normally executes ABORT, but may be altered (with care) to a user's alternative procedure.
#	d1 --- d2 Generate from a double number d1, the next ascii character which is placed in an output string. Result d2 is the quotient after division by BASE, and is maintained for further processing. Used between <# and #>. See #S.	L0	(DO)	The run-time procedure compiled by DO which moves the loop control parameters to the return stack. See DO.
#>	d --- addr count Terminates numeric output conversion by dropping d, leaving the text address and character count suitable for TYPE.	L0	(FIND)	addr1 addr2 --- pfa b tf (ok) addr1 addr2 --- ff (bad) Searches the dictionary starting at the name field address addr2, matching to the text at addr1. Returns parameter field address, length byte of name field and boolean true for a good match. If no match is found, only a boolean false is left.
#S	d1 --- d2 Generates ascii text in the text output buffer, by the use of #, until a zero double number n2 results. Used between <# and #>.	L0	(LINE)	n1 n2 --- addr count Convert the line number n1 and the screen n2 to the disc buffer address containing the data. A count of 64 indicates the full line text length.
	--- addr Used in the form: nnnn Leaves the parameter field address of dictionary word nnnn. As a compiler directive, executes in a colon-definition to compile the address as a literal. If the word is not found after a search of CONTEXT and CURRENT, an appropriate error message is given. Pronounced "tick".	P,L0	(LOOP)	The run-time procedure compiled by LOOP which increments the loop index and tests for loop completion. See LOOP.
(Used in the form: (cccc) Ignore a comment that will be delimited by a right parenthesis on the same line. May occur during execution or in a colon-definition. A blank after the leading parenthesis is required.	P,L0	(NUMBER)	d1 addr1 --- d2 addr2 Convert the ascii text beginning at addr1+1 with regard to BASE. The new value is accumulated into double number d1, being left as d2. Addr2 is the address of the first unconvertable digit. Used by NUMBER.
(.)	The run-time procedure, compiled by . which transmits the following in-line text to the selected output device. See .	C+	*	n1 n2 --- prod Leave the signed product of two signed numbers.
(;CODE)	The run-time procedure, compiled by ;CODE, that rewrites the code field of the most recently defined word to point to the following machine code sequence. See ;CODE.	C	*/	n1 n2 n3 --- n4 Leave the ratio n4 = n1*n2/n3 where all are signed numbers. Retention of an intermediate 31 bit product permits greater accuracy than would be available with the sequence: n1 n2 * n3 /
			*/MOD	n1 n2 n3 --- n4 n5 Leave the quotient n5 and remainder n4 of the operation n1*n2/n3. A 31 bit intermediate product is used as for */.

FORTH INTEREST GROUP P.O. Box 1105 San Carlos, Ca. 94070

FORTH INTEREST GROUP P.O. Box 1105 San Carlos, Ca. 94070

?ERROR	<code>f n ---</code> Issue an error message number <code>n</code> , if the boolean flag is true.	B/BUF	<code>--- n</code> This constant leaves the number of bytes per disc buffer, the byte count read from disc by BLOCK .
?EXEC	Issue an error message if not executing.	B/SCR	<code>--- n</code> This constant leaves the number of blocks per editing screen. By convention, an editing screen is 1024 bytes organized as 16 lines of 64 characters each.
?LOADING	Issue an error message if not loading		
?PAIRS	<code>n1 n2 ---</code> Issue an error message if <code>n1</code> does not equal <code>n2</code> . The message indicates that compiled conditionals do not match.	BACK	<code>addr ---</code> Calculate the backward branch offset from HERE to <code>addr</code> and compile into the next available dictionary memory address.
?STACK	Issue an error message if the stack is out of bounds. This definition may be installation dependent.	BASE	<code>--- addr</code> U,LO A user variable containing the current number base used for input and output conversion.
?TERMINAL	<code>--- f</code> Perform a test of the terminal keyboard for actuation of the break key. A true flag indicates actuation. This definition is installation dependent.	BEGIN	<code>--- addr n (compiling) P,LO</code> Occurs in a colon-definition in form: <code>BEGIN ... UNTIL</code> <code>BEGIN ... AGAIN</code> <code>BEGIN ... WHILE ... REPEAT</code> At run-time, BEGIN marks the start of a sequence that may be repetitively executed. It serves as a return point from the corresponding UNTIL , AGAIN or REPEAT . When executing UNTIL , a return to BEGIN will occur if the top of the stack is false; for AGAIN and REPEAT a return to BEGIN always occurs. At compile time BEGIN leaves its return address and <code>n</code> for compiler error checking.
@	<code>addr --- n</code> LO Leave the 16 bit contents of address.		
ABORT	Clear the stacks and enter the execution state. Return control to the operators terminal, printing a message appropriate to the installation.		
ABS	<code>n --- u</code> LO Leave the absolute value of <code>n</code> as <code>u</code> .	BL	<code>--- c</code> A constant that leaves the ascii value for "blank".
AGAIN	<code>addr n --- (compiling) P,C2,LO</code> Used in a colon-definition in the form: <code>BEGIN ... AGAIN</code> At run-time, AGAIN forces execution to return to corresponding BEGIN . There is no effect on the stack. Execution cannot leave this loop (unless R> DROP is executed one level below). At compile time, AGAIN compiles BRANCH with an offset from HERE to <code>addr</code> . <code>n</code> is used for compile-time error checking.	BLANKS	<code>addr count ---</code> Fill an area of memory beginning at <code>addr</code> with blanks.
ALLOT	<code>n ---</code> LO Add the signed number to the dictionary pointer DP . May be used to reserve dictionary space or re-origin memory. <code>n</code> is with regard to computer address type (byte or word).	BLK	<code>--- addr</code> U,LO A user variable containing the block number being interpreted. If zero, input is being taken from the terminal input buffer.
AND	<code>n1 n2 --- n2</code> LO Leave the bitwise logical and of <code>n1</code> and <code>n2</code> as <code>n3</code> .	BLOCK	<code>n --- addr</code> LO Leave the memory address of the block buffer containing block <code>n</code> . If the block is not already in memory, it is transferred from disc to which ever buffer was least recently written. If the block occupying that buffer has been marked as updated, it is re-written to disc before block <code>n</code> is read into the buffer. See also BUFFER , R/W UPDATE FLUSH

FORTH INTEREST GROUP P.O. Box 1105 San Carlos, Ca. 94070

10

BLOCK-READ	These are the preferred names for the installation dependent code to read and write one block to the disc.	COMPILE	When the word containing COMPILE executes, the execution address of the word following COMPILE is copied (compiled) into the dictionary. This allows specific compilation situations to be handled in addition to simply compiling an execution address (which the interpreter already does).	C2
BRANCH	The run-time procedure to unconditionally branch. An in-line offset is added to the interpretive pointer IP to branch ahead or back. BRANCH is compiled by ELSE, AGAIN, REPEAT.	CONSTANT	n --- A defining word used in the form: n CONSTANT cccc to create word cccc, with its parameter field containing n. When cccc is later executed, it will push the value of n to the stack.	L0
BUFFER	n --- addr Obtain the next memory buffer, assigning it to block n. If the contents of the buffer is marked as updated, it is written to the disc. The block is not read from the disc. The address left is the first cell within the buffer for data storage.	CONTEXT	--- addr A user variable containing a pointer to the vocabulary within which dictionary searches will first begin.	U,L0
C!	b addr --- Store 8 bits at address. On word addressing computers, further specification is necessary regarding byte addressing.	COUNT	addr1 --- addr2 n Leave the byte address addr2 and byte count n of a message text beginning at address addr1. It is presumed that the first byte at addr1 contains the text byte count and the actual text starts with the second byte. Typically COUNT is followed by TYPE.	L0
C.	b --- Store 8 bits of b into the next available dictionary byte, advancing the dictionary pointer. This is only available on byte addressing computers, and should be used with caution on byte addressing mini-computers.	CR	Transmit a carriage return and line feed to the selected output device.	L0
C@	addr --- b Leave the 8 bit contents of memory address. On word addressing computers, further specification is needed regarding byte addressing.	CREATE	A defining word used in the form: CREATE cccc by such words as CODE and CONSTANT to create a dictionary header for a Forth definition. The code field contains the address of the words parameter field. The new word is created in the CURRENT vocabulary.	
CFA	pfa --- cfa Convert the parameter field address of a definition to its code field address.	CSP	---- addr A user variable temporarily storing the stack pointer position, for compilation error checking.	U
CMOVE	from to count --- Move the specified quantity of bytes beginning at address from to address to. The contents of address from is moved first proceeding toward high memory. Further specification is necessary on word addressing computers.	D+	d1 d2 --- dsun Leave the double number sum of two double numbers.	
COLD	The cold start procedure to adjust the dictionary pointer to the minimum standard and restart via ABORT. May be called from the terminal to remove application programs and restart.	D+-	d1 n --- d2 Apply the sign of n to the double number d1, leaving it as d2.	
		D.	d --- Print a signed double number from a 32 bit two's complement value. The high-order 16 bits are most accessible on the stack. Conversion is performed according to the current BASE. A blank follows. Pronounced D-dot.	L1

FORTH INTEREST GROUP P.O. Box 1105 San Carlos, Ca. 94070 //

D.R	d n --- Print a signed double number d right aligned in a field n characters wide.	DO	n1 n2 --- (execute) addr n --- (compile) P,C2;LO Occurs in a colon-definition in form: DO ... LOOP DO ... +LOOP At run time, DO begins a sequence with repetitive execution controlled by a loop limit n1 and an index with initial value n2. DO removes these from the stack. Upon reaching LOOP the index is incremented by one. Until the new index equals or exceeds the limit, execution loops back to just after DO; otherwise the loop parameters are discarded and execution continues ahead. Both n1 and n2 are determined at run-time and may be the result of other operations. Within a loop 'I' will copy the current value of the index to the stack. See I, LOOP, +LOOP, LEAVE. When compiling within the colon-definition, DO compiles (DO), leaves the following address addr and n for later error checking.
DABS	d --- ud Leave the absolute value ud of a double number.		
DECIMAL	Set the numeric conversion BASE for decimal input-output.	LO	
DEFINITIONS	Used in the form: cccc DEFINITIONS Set the CURRENT vocabulary to the CONTEXT vocabulary. In the example, executing vocabulary name cccc made it the CONTEXT vocabulary and executing DEFINITIONS made both specify vocabulary cccc.	L1	
DIGIT	c n1 --- n2 tf (ok) c n1 --- ff (bad) Converts the ascii character c (using base n1) to its binary equivalent n2, accompanied by a true flag. If the conversion is invalid, leaves only a false flag.	DOES>	LO A word which defines the run-time action within a high-level defining word. DOES> alters the code field and first parameter of the new word to execute the sequence of compiled word addresses following DOES>. Used in combination with <BUILDS. When the DOES> part executes it begins with the address of the first parameter of the new word on the stack. This allows interpretation using this area or its contents. Typical uses include the Forth assembler, multi-dimensional arrays, and compiler generation.
DLIST	List the names of the dictionary entries in the CONTEXT vocabulary.		
DLITERAL	d --- d (executing) d --- (compiling) P If compiling, compile a stack double number into a literal. Later execution of the definition containing the literal will push it to the stack. If executing, the number will remain on the stack.	DP	---- addr U,L A user variable, the dictionary pointer, which contains the address of the next free memory above the dictionary. The value may be read by HERE and altered by ALLOT.
DMINUS	d1 --- d2 Convert d1 to its double number two's complement.	DPL	---- addr U,L A user variable containing the number of digits to the right of the decimal on double integer input. It may also be used hold output column location of a decimal point, in user generated formatting. The default value on single number input is -1.
		DRO DRI	Installation dependent commands to select disc drives, by presetting OFFSET. The contents of OFFSET is added to the block number in BLOCK to allow for this selection. Offset is suppressed for error text so that it may always originate from drive 0.

DROP	n --- Drop the number from the stack.	LO	ENDIF	addr n --- (compile) P,C0,LO Occurs in a colon-definition in form: IF ... ENDIF IF ... ELSE ... ENDIF
DUMP	addr n --- Print the contents of a memory locations beginning at addr. Both addresses and contents are shown in the current numeric base.	LO		At run-time, ENDIF serves only as the destination of a forward branch from IF or ELSE. It marks the conclusion of the conditional structure. THEN is another name for ENDIF. Both names are supported in fig-FORTH. See also IF and ELSE.
DUP	n --- n n Duplicate the value on the stack.	LO		At compile-time, ENDIF computes the forward branch offset from addr to HERE and stores it at addr. n is used for error tests.
ELSE	addr1 n1 --- addr2 n2 (compiling) P,C2,LO Occurs within a colon-definition in the form: IF ... ELSE ... ENDIF At run-time, ELSE executes after the true part following IF. ELSE forces execution to skip over the following false part and resumes execution after the ENDIF. It has no stack effect. At compile-time ELSE emplaces BRANCH reserving a branch offset, leaves the address addr2 and n2 for error testing. ELSE also resolves the pending forward branch from IF by calculating the offset from addr1 to HERE and storing at addr1.		ERASE	addr n --- Clear a region of memory to zero from addr over n addresses.
EMIT	c --- Transmit ascii character c to the selected output device. OUT is incremented for each character output.	LO	ERROR	line --- in blk Execute error notification and re-start of system. WARNING is first examined. If 1, the text of line n, relative to screen 4 of drive 0 is printed. This line number may be positive or negative, and beyond just screen 4. If WARNING=0, n is just printed as a message number (non disc installation). If WARNING is -1, the definition (ABORT) is executed, which executes the system ABORT. The user may cautiously modify this execution by altering (ABORT). fig-FORTH saves the contents of IN and BLK to assist in determining the location of the error. Final action is execution of QUIT.
EMPTY-BUFFERS	 Mark all block-buffers as empty, not necessarily affecting the contents. Updated blocks are not written to the disc. This is also an initialization procedure before first use of the disc.	LO	EXECUTE	addr -- Execute the definition whose code field address is on the stack. The code field address is also called the compilation address.
ENCLOSE	addr1 c --- addr1 n1 n2 n3 The text scanning primitive used by WORD. From the text address addr1 and an ascii delimiting character c, is determined the byte offset to the first non-delimiter character n1, the offset to the first delimiter after the text n2, and the offset to the first character not included. This procedure will not process past an ascii 'null', treating it as an unconditional delimiter.		EXPECT	addr count --- LO Transfer characters from the terminal to address, until a "return" or the count of characters have been received. One or more nulls are added at the end of the text.
END	P,C2,LO This is an 'alias' or duplicate definition for UNTIL.		FENCE	--- addr U A user variable containing an address below which FORGETting is trapped. To forget below this point the user must alter the contents of FENCE.
			FILL	addr quan b --- Fill memory at the address with the specified quantity of bytes b.
			FIRST	--- n A constant that leaves the address of the first (lowest) block buffer.

FLD	--- addr	U	IF	f --- (run-time) --- addr n (compile) P,C2,L0 Occurs is a colon-definition in form: IF (tp) ... ENDIF IF (tp) ... ELSE (fp) ... ENDIF At run-time, IF selects execution based on a boolean flag. If f is true (non-zero), execution continues ahead thru the true part. If f is false (zero), execution skips till just after ELSE to execute the false part. After either part, execution resumes after ENDIF. ELSE and its false part are optional; if missing, false execution skips to just after ENDIF.
FORGET	Executed in the form: FORGET cccc Deletes definition named cccc from the dictionary with all entries physically following it. In fig-FORTH, an error message will occur if the CURRENT and CONTEXT vocabularies are not currently the same.	E,L0		
FORTH	The name of the primary vocabulary. Execution makes FORTH the CONTEXT vocabulary. Until additional user vocabularies are defined, new user definitions become a part of FORTH. FORTH is immediate, so it will execute during the creation of a colon-definition, to select this vocabulary at compile time.	P,L1		At compile-time IF compiles OBRANCH and reserves space for an offset at addr. addr and n are used later for resolution of the offset and error testing.
HERE	--- addr	L0	IMMEDIATE	Mark the most recently made definition so that when encountered at compile time, it will be executed rather than being compiled. i.e. the precedence bit in its header is set. This method allows definitions to handle unusual compiling situations, rather than build them into the fundamental compiler. The user may force compilation of an immediate definition by preceding it with [COMPILE].
HEX	Set the numeric conversion base to sixteen (hexadecimal).	L0		
HL0	--- addr	L0	IN	--- addr L0 A user variable containing the byte offset within the current input text buffer (terminal or disc) from which the next text will be accepted. WORD uses and moves the value of IN.
HOLD	c --- Used between <# and #> to insert an ascii character into a pictured numeric output string. e.g. 2# HOLD will place a decimal point.	L0	INDEX	from to --- Print the first line of each screen over the range from, to. This is used to view the comment lines of an area of text on disc screens.
I	--- n	C,L0	INTERPRET	The outer text interpreter which sequentially executes or compiles text from the input stream (terminal or disc) depending on STATE. If the word name cannot be found after a search of CONTEXT and then CURRENT it is converted to a number according to the current base. That also failing, an error message echoing the name with a " ? " will be given. Text input will be taken according to the convention for WORD. If a decimal point is found as part of a number, a double number value will be left. The decimal point has no other purpose than to force this action. See NUMBER.
ID.	addr --- Print a definition's name from its name field address.			

KEY	--- c	L0 LOOP	addr n --- (compiling) P,C2,L0 Occurs in a colon-definition in form: DO ... LOOP At run-time, LOOP selectively controls branching back to the corresponding DO based on the loop index and limit. The loop index is incremented by one and compared to the limit. The branch back to DO occurs until the index equals or exceeds the limit; at that time, the parameters are discarded and execution continues ahead.
LATEST	--- addr		
	Leave the name field address of the topmost word in the CURRENT vocabulary.		
LEAVE		C,L0	
	Force termination of a DO-LOOP at the next opportunity by setting the loop limit equal to the current value of the index. The index itself remains unchanged, and execution proceeds normally until LOOP or +LOOP is encountered.		At compile-time, LOOP compiles (LOOP) and uses addr to calculate an offset to DO. n is used for error testing.
LFA	pfa --- lfa	M*	n1 n2 --- d A mixed magnitude math operation which leaves the double number signed product of two signed number.
	Convert the parameter field address of a dictionary definition to its link field address.		
LIMIT	---- n	M/	d n1 --- n2 n3 A mixed magnitude math operator which leaves the signed remainder n2 and signed quotient n3, from a double number dividend and divisor n1. The remainder takes its sign from the dividend.
	A constant leaving the address just above the highest memory available for a disc buffer. Usually this is the highest system memory.		
LIST	n ---	L0 M/MOD	ud1 u2 --- u3 ud4 An unsigned mixed magnitude math operation which leaves a double quotient ud4 and remainder u3, from a double dividend ud1 and single divisor u2.
	Display the ascii text of screen n on the selected output device. SCR contains the screen number during and after this process.		
LIT	--- n	C2,L0	
	Within a colon-definition, LIT is automatically compiled before each 16 bit literal number encountered in input text. Later execution of LIT causes the contents of the next dictionary address to be pushed to the stack.	MAX	n1 n2 --- max Leave the greater of two numbers.
LITERAL	n --- (compiling) P,C2,L0	MESSAGE	n --- Print on the selected output device the text of line n relative to screen 4 of drive 0. n may be positive or negative. MESSAGE may be used to print incidental text such as report headers. If WARNING is zero, the message will simply be printed as a number (disc un-available).
	If compiling, then compile the stack value n as a 16 bit literal. This definition is immediate so that it will execute during a colon definition. The intended use is: : xxx [calculate] LITERAL ; Compilation is suspended for the compile time calculation of a value. Compilation is resumed and LITERAL compiles this value.	MIN	n1 n2 --- min Leave the smaller of two numbers.
		MINUS	n1 --- n2 Leave the two's complement of a number.
LOAD	n ---	L0 MOD	n1 n2 --- mod Leave the remainder of n1/n2, with the same sign as n1.
	Begin interpretation of screen n. Loading will terminate at the end of the screen or at ;S. See ;S and -->.	MON	Exit to the system monitor, leaving a re-entry to Forth, if possible.

MOVE <code>addr1 addr2 n ---</code> Move the contents of n memory cells (16 bit contents) beginning at <code>addr1</code> into n cells beginning at <code>addr2</code> . The contents of <code>addr1</code> is moved first. This definition is appropriate on on word addressing computers.	PAD <code>--- addr</code> L0 Leave the address of the text output buffer, which is a fixed offset above HERE .
NEXT This is the inner interpreter that uses the interpretive pointer IP to execute compiled Forth definitions. It is not directly executed but is the return point for all code procedures. It acts by fetching the address pointed by IP, storing this value in register W. It then jumps to the address pointed to by the address pointed to by W. W points to the code field of a definition which contains the address of the code which executes for that definition. This usage of indirect threaded code is a major contributor to the power, portability, and extensibility of Forth. Locations of IP and W are computer specific.	PFA <code>nfa --- pfa</code> Convert the name field address of a compiled definition to its parameter field address.
NFA <code>pfa --- nfa</code> Convert the parameter field address of a definition to its name field.	POP The code sequence to remove a stack value and return to NEXT . POP is not directly executable, but is a Forth re-entry point after machine code.
NUMBER <code>addr --- d</code> Convert a character string left at <code>addr</code> with a preceeding count, to a signed double number, using the current numeric base. If a decimal point is encountered in the text, its position will be given in DPL, but no other effect occurs. If numeric conversion is not possible, an error message will be given.	PREV <code>---- addr</code> A variable containing the address of the disc buffer most recently referenced. The UPDATE command marks this buffer to be later written to disc.
OFFSET <code>--- addr</code> U A user variable which may contain a block offset to disc drives. The contents of OFFSET is added to the stack number by BLOCK . Messages by MESSAGE are independent of OFFSET . See BLOCK , DRO , DRI , MESSAGE .	PUSH This code sequence pushes machine registers to the computation stack and returns to NEXT . It is not directly executable, but is a Forth re-entry point after machine code.
OR <code>n1 n2 -- or</code> L0 Leave the bit-wise logical or of two 16 bit values.	PUT This code sequence stores machine register contents over the topmost computation stack value and returns to NEXT . It is not directly executable, but is a Forth re-entry point after machine code.
OUT <code>--- addr</code> U A user variable that contains a value incremented by EMIT . The user may alter and examine OUT to control display formatting.	QUERY Input 80 characters of text (or until a "return") from the operators terminal. Text is positioned at the address contained in TIB with IN set to zero.
OVER <code>n1 n2 --- n1 n2 n1</code> L0 Copy the second stack value, placing it as the new top.	QUIT L1 Clear the return stack, stop compilation, and return control to the operators terminal. No message is given.
	R <code>--- n</code> Copy the top of the return stack to the computation stack.
	R# <code>--- addr</code> U A user variable which may contain the location of an editing cursor, or other file related function.

R/W	addr blk f --- The fig-FORTH standard disc read-write linkage. addr specifies the source or destination block buffer, blk is the sequential number of the referenced block; and f is a flag for f=0 write and f=1 read. R/W determines the location on mass storage, performs the read-write and performs any error checking.		
R>	--- n Remove the top value from the return stack and leave it on the computation stack. See >R and R.	LO	
RO	--- addr A user variable containing the initial location of the return stack. Pronounced R-zero. See RPI	U	
REPEAT	addr n --- (compiling) P,C2 Used within a colon-definition in the form: BEGIN ... WHILE ... REPEAT At run-time, REPEAT forces an unconditional branch back to just after the corresponding BEGIN. At compile-time, REPEAT compiles BRANCH and the offset from HERE to addr. n is used for error testing.		
ROT	n1 n2 n3 --- n2 n3 n1 Rotate the top three values on the stack, bringing the third to the top.	LO	
RPI	A computer dependent procedure to initialize the return stack pointer from user variable RO.		
S->D	n --- d Sign extend a single number to form a double number.		
SO	--- addr A user variable that contains the initial value for the stack pointer. Pronounced S-zero. See SPI	U	
SCR	--- addr A user variable containing the screen number most recently reference by LIST.	U	
SIGN	n d --- d Stores an ascii "-" sign just before a converted numeric output string in the text output buffer when n is negative. n is discarded, but double number d is maintained. Must be used between <# and #>.	LO	
			SMUDGE Used during word definition to toggle the "smudge bit" in a definitions name field. This prevents an un-completed definition from being found during dictionary searches, until compiling is completed without error.
			SPI A computer dependent procedure to initialize the stack pointer from SO.
			SP@ --- addr A computer dependent procedure to return the address of the stack position to the top of the stack, as it was before SP@ was executed. (e.g. 1 2 SP@ @ . . . would type 2 2 1)
			SPACE LO Transmit an ascii blank to the output device.
			SPACES n --- LO Transmit n ascii blanks to the output device.
			STATE --- addr LO,U A user variable containing the compilation state. A non-zero value indicates compilation. The value itself may be implementation dependent.
			SWAP n1 n2 --- n2 n1 LO Exchange the top two values on the stack.
			TASK A no-operation word which can mark the boundary between applications. By forgetting TASK and re-compiling, an application can be discarded in its entirety.
			THEN P,C,O,LO An alias for ENDIF.
			TIB --- addr U A user variable containing the address of the terminal input buffer.
			TOGGLE addr b --- Complement the contents of addr by the bit pattern b.
			TRAVERSE addr1 n --- addr2 Move across the name field of a fig-FORTH variable length name field. addr1 is the address of either the length byte or the last letter. If n=1, the motion is toward hi memory; if n=-1, the motion is toward low memory. The addr2 resulting is address of the other end of the name.

FORTH INTEREST GROUP P.O. Box 1105 San Carlos, Ca. 94070

17

TRIAD	<pre>scr --- Display on the selected output device the three screens which include that numbered scr, beginning with a screen evenly divisible by three. Output is suitable for source text records, and includes a reference line at the bottom taken from line 15 of screen4.</pre>	<p>VARIABLE E,LU</p> <p>A defining word used in the form: <pre>n VARIABLE cccc</pre> When VARIABLE is executed, it creates the definition cccc with its parameter field initialized to n. When cccc is later executed, the address of its parameter field (containing n) is left on the stack, so that a fetch or store may access this location.</p>
TYPE	<pre>addr count --- LO Transmit count characters from addr to the selected output device.</pre>	<p>VOC-LINK U</p> <p>--- addr</p> <p>A user variable containing the address of a field in the definition of the most recently created vocabulary. All vocabulary names are linked by these fields to allow control for FORGETting thru multiple vocabularys.</p>
U*	<pre>u1 u2 --- ud Leave the unsigned double number product of two unsigned numbers.</pre>	
U/	<pre>ud u1 --- u2 u3 Leave the unsigned remainder u2 and unsigned quotient u3 from the unsigned double dividend ud and unsigned divisor u1.</pre>	<p>VOCABULARY E,L</p> <p>A defining word used in the form: <pre>VOCABULARY cccc</pre> to create a vocabulary definition cccc. Subsequent use of cccc will make it the CONTEXT vocabulary which is searched first by INTERPRET. The sequence "cccc DEFINITIONS" will also make cccc the CURRENT vocabulary into which new definitions are placed.</p> <p>In fig-FORTH, cccc will be so chained as to include all definitions of the vocabulary in which cccc is itself defined. All vocabularys ultimately chain to Forth. By convention, vocabulary names are to be declared IMMEDIATE. See VOC-LINK.</p>
UNTIL	<pre>f --- (run-time) addr n --- (compile) P,C2,LO Occurs within a colon-definition in the form: BEGIN ... UNTIL At run-time, UNTIL controls the conditional branch back to the corresponding BEGIN. If f is false, execution returns to just after BEGIN; if true, execution continues ahead. At compile-time, UNTIL compiles (OBRANCH) and an offset from HERE to addr. n is used for error tests.</pre>	<p>VLIST</p> <p>List the names of the definitions in the context vocabulary. "Break" will terminate the listing.</p>
UPDATE	<pre>LO Marks the most recently referenced block (pointed to by PREV) as altered. The block will subsequently be transferred automatically to disc should its buffer be required for storage of a different block.</pre>	<p>WARNING U</p> <p>--- addr</p> <p>A user variable containing a value controlling messages. If = 1 disc is present, and screen 4 of drive 0 is the base location for messages. If = 0, no disc is present and messages will be presented by number. If = -1, execute (ABORT) for a user specified procedure. See MESSAGE, ERROR.</p>
USE	<pre>--- addr A variable containing the address of the block buffer to use next, as the least recently written.</pre>	
USER	<pre>n --- LO A defining word used in the form: n USER cccc which creates a user variable cccc. The parameter field of cccc contains n as a fixed offset relative to the user pointer register UP for this user variable. When cccc is later executed, it places the sum of its offset and the user area base address on the stack as the storage address of that particular variable.</pre>	<p>WHILE</p> <pre>f --- (run-time) ad1 n1 --- ad1 n1 ad2 n2 P,C2 Occurs in a colon-definition in the form: BEGIN ... WHILE (tp) ... REPEAT At run-time, WHILE selects conditional execution based on boolean flag f. If f is true (non-zero), WHILE continues execution of the true part thru to REPEAT, which then branches back to BEGIN. If f is false (zero), execution skips to just after REPEAT, exiting the structure. At compile time, WHILE emplaces (OBRANCH) and leaves ad2 of the reserved offset. The stack values will be resolved by REPEAT.</pre>

```

WIDTH      --- addr      U
In fig-FORTH, a user variable containing the maximum number of letters saved in the compilation of a definitions' name. It must be 1 thru 31, with a default value of 31. The name character count and its natural characters are saved, up to the value in WIDTH. The value may be changed at any time within the above limits.

```

WORD **c** **---** **LO**

Read the next text characters from the input stream being interpreted, until a delimiter **c** is found, storing the packed character string beginning at the dictionary buffer **HERE**. **WORD** leaves the character count in the first byte, the characters, and ends with two or more blanks. Leading occurrences of **c** are ignored. If **BLK** is zero, text is taken from the terminal input buffer, otherwise from the disc block stored in **BLK**. See **BLK**. **IN**.

X This is pseudonym for the "null"
or dictionary entry for a name of
one character of ascii null. It
is the execution procedure to term-
inate interpretation of a line of
text from the terminal or within
a disc buffer, as both buffers always
have a null at the end.

```
XOR      n1 n2 --- xor      L1
Leave the bitwise logical exclusive-
or of two values.
```

```

[      P, L1
      Used in a colon-definition in form:
      : xxx [ words ] more ;
      Suspend compilation. The words after
      [ are executed, not compiled. This
      allows calculation or compilation
      exceptions before resuming compilation
      with ]. See LITERAL. ].

```

```
[COMPILE]                                P,C
Used in a colon-definition in form:
: xxx [COMPILE] FORTH ;
[COMPILE] will force the compilation
of an immediate definition,
that would otherwise execute
during compilation. The above
example will select the FORTH
vocabulary when xxx executes, rather
than at compile time.
```

1 Resume compilation, to the completion L1
of a colon-definition. See [.

FORTH INTEREST GROUP P.O. Box 1105 San Carlos, Ca. 94070 19

FORTH — 79

A PUBLICATION OF THE FORTH STANDARDS TEAM

October 1980

Produced by: FORTH Standards Team

Distributed by: FORTH Interest Group

FORTH INTEREST GROUP ----- P.O. Box 1105 ----- San Carlos, CA 94070

FORTH-79 STANDARD

A PUBLICATION OF THE FORTH STANDARDS TEAM

CONTENTS

	<u>Page</u>
0. FOREWORD	1
1. PURPOSE	1
2. SCOPE	1
3. ORGANIZATION	1
4. DEFINITION OF TERMS	2
5. REFERENCES	9
6. REQUIREMENTS	10
7. COMPLIANCE AND LABELING	11
8. USE	12
9. GLOSSARY NOTATION	13
10. REQUIRED WORD SET	15
11. EXTENSION WORD SETS	32
11.1 Double Number Word Set	32
11.2 Assembler Word Set	34
12. EXPERIMENTAL PROPOSALS	34
REFERENCE WORD SET	
FORTH-79 HANDY REFERENCE CARD	

FORTH-79 STANDARD**A PUBLICATION OF THE FORTH STANDARDS TEAM****O. FOREWORD**

The computer language FORTH was created by Mr. Charles Moore, as an extensible, multi-level environment containing elements of an operating system, a machine monitor, and facilities for program development and testing.

This Standard is a direct descendant of FORTH-77, a work of the FORTH Users Group (Europe). The constituency of the Standards Team has steadily broadened, to include users of an increasing variety of host computers.

1. PURPOSE

The purpose of this FORTH Standard is to allow transportability of standard FORTH programs in source form among standard FORTH systems. A standard program shall execute equivalently on all standard FORTH systems.

2. SCOPE

This standard shall apply to any Standard FORTH program executing on any Standard FORTH system, provided sufficient computer resources (memory, mass storage) are available.

3. ORGANIZATION

This standard consists of:

- 1) General Text
- 2) Definitions of Terms
- 3) Required Word Set
- 4) Extension Word Sets

Word sets may be subdivided for conceptual purposes by function:

Nucleus
Interpreter
Compiler
Devices

Tradeoffs

When conflicting choices must be made, the following order shall guide the Standards Team.

- 1) Functional correctness
 - known bounds, non-ambiguous.
- 2) Portability
 - repeatable results when transported among Standard systems.
- 3) Simplicity.
- 4) Naming clarity
 - uniformity of expression. Descriptive names are preferred over procedural. (i.e., [COMPILE] rather than 'C, and ALLOT rather than DP+! .)
- 5) Generality.
- 6) Execution speed.
- 7) Memory compactness.
- 8) Compilation speed.
- 9) Historical continuity.
- 10) Pronounceability.
- 11) Teachability.

4. DEFINITIONS OF TERMS

These definitions, when in lower case, are terms used within this Standard. They present terms as specifically used within FORTH.

address, byte

An unsigned number that locates an 8-bit byte in a standard FORTH address space over {0..65,535}. It may be a native machine address or a representation on a virtual machine, locating the 'addr-th' byte within the virtual byte address space. Address arithmetic is modulo 65,536 without overflow.

address, compilation

The numerical value equivalent to a FORTH word definition, which is compiled for that definition. The address interpreter uses this value to locate the machine code corresponding to each definition. (May also be called the code field address.)

address, native machine

The natural address representation of the host computer.

address, parameter field

The address of the first byte of memory associated with a word definition for the storage of compilation addresses (in a colon-definition), numeric data and text characters.

arithmetic

All integer arithmetic is performed with signed 16 or 32 bit two's complement results, unless noted.

block

The unit of data from mass storage, referenced by block number. A block must contain 1024 bytes regardless of the minimum data unit read/written from mass storage. The translation from block number to device and physical record is a function of the implementation.

block buffer

A memory area where a mass storage block is maintained.

byte

An assembly of 8 bits. In reference to memory, it is the storage capacity for 8 bits.

cell

A 16-bit memory location. The n-th cell contains the 2n-th and (2n+1)-th byte of the FORTH address space. The byte order is presently unspecified.

character

A 7-bit number which represents a terminal character. The ASCII character set is considered standard. When contained in a larger field, the higher order bits are zero.

compilation

The action of accepting text words from the input stream and placing corresponding compilation addresses in a new dictionary entry.

defining word

A word that, when executed, creates a new dictionary entry. The new word name is taken from the input stream. If the input stream is exhausted before the new name is available, an error condition exists. Common

defining words are:

: CONSTANT CREATE

definition

See 'word definition'.

dictionary

A structure of word definitions in a computer memory. In systems with a text interpreter, the dictionary entries are organized in vocabularies to enable location by name. The dictionary is extensible, growing toward high memory.

equivalent execution

For the execution of a standard program, a set of non-time dependent inputs will produce the same non-time dependent outputs on any FORTH Standard System with sufficient resources to execute the program. Only standard source code will be transportable.

error condition

An exceptional condition which requires action by the system other than the expected function. Actions may be:

1. ignore, and continue
2. display a message
3. execute a particular word
4. interpret a block
5. return control to the text interpreter

A Standard System shall be provided with a tabulation of the action taken for all specified error conditions.

General error conditions:

1. input stream exhausted before a required <name>.
2. empty stack and full stack for the text interpreter.
3. an unknown word, not a valid number for the text interpreter.
4. compilation of incorrectly nested conditionals.
5. interpretation of words restricted to compilation.

6. FORGETing within the system to a point that removes a word required for correct execution.
7. insufficient space remaining in the dictionary.

false

A zero number represents the false condition flag.

flag

A number that may have two logical states, zero and non-zero. These are named 'true' = non-zero, and 'false' = zero. Standard word definitions leave 1 for true, 0 for false.

glossary

A set of word definitions given in a natural language describing the corresponding computer execution action.

immediate word

A word defined to automatically execute when encountered during compilation, which handles exception cases to the usual compilation. See IF LITERAL ." etc.

input stream

A sequence of characters available to the system, for processing by the text interpreter. The input stream conventionally may be taken from a terminal (via the terminal input buffer) and mass storage (via a block buffer). >IN and BLK specify the input stream. Words using or altering >IN and BLK are responsible for maintaining and restoring control of the input stream.

interpreter, address

The (set of) word definitions which interprets (sequences of) FORTH compilation addresses by executing the word definition specified for each one.

interpreter, text

The (set of) word definitions that repeatedly accepts a word name from the input stream, locates the corresponding dictionary entry, and starts the address interpreter to execute it. Text in the input stream interpreted as a number leaves the corresponding value on the data stack. When in the compile mode, the addresses of FORTH words are compiled into the dictionary for later interpretation by the address interpreter. In this case, numbers are compiled, to be placed on the data stack when later interpreted. Numbers shall be accepted unsigned or negatively signed, according to BASE.

load

The acceptance of text from a mass storage device and execution of the dictionary definition of the words encountered. This is the general method for compilation of new definitions into the dictionary.

mass storage

Data is read from mass storage in the form of 1024 byte blocks. This data is held in block buffers. When indicated as UPDATED (modified) data will be ultimately written to mass storage.

number

When values exist within a larger field, the high order bits are zero. When stored in memory the byte order of a number is unspecified.

<u>type</u>	<u>range</u>	<u>minimum field</u>
bit	0..1	1
character	0..127	7
byte	0..255	8
number	-32,768..32,767	16
positive number	0..32,767	16
unsigned number	0..65,535	16
double number	-2,147,483,648.. 2,147,483,647	32
positive double number	0..2,147,483,647	32
unsigned double number	0..4,294,967,295	32

When represented on the stack, the higher 16-bits (with sign) of a double number are most accessible. When in memory the higher 16-bits are at the lower address. Storage extends over four bytes toward high memory. The byte order within each 16-bit field is unspecified.

output, pictured

The use of numeric output primitives, which convert numerical values into text strings. The operators are used in a sequence which resembles a symbolic 'picture' of the desired text format. Conversion proceeds from low digit to high, from high memory to low.

program

A complete specification of execution to achieve a specific function (application task) expressed in FORTH source code form.

return

The means of terminating text from the input stream. (Conventionally a null (ASCII 0) indicates end of text in the input stream. This character is left by the 'return' key actuation of the operator's terminal, as an absolute stopper to text interpretation.)

screen

Textual data arranged for editing. By convention, a screen consists of 16 lines (numbered 0 thru 15) of 64 characters each. Screens usually contain program source text, but may be used to view mass storage data. The first byte of a screen occupies the first byte of a mass storage block, which is the beginning point for text interpretation during a load.

source definition

Text consisting of word names suitable for execution by the text interpreter. Such text is usually arranged in screens and maintained on a mass storage device.

stack, data

A last in, first out list consisting of 16-bit binary values. This stack is primarily used to hold intermediate values during execution of word definitions. Stack values may represent numbers, characters, addresses, boolean values, etc.

When the name 'stack' is used, it implies the data stack.

stack, return

A last in, first out list which contains the machine addresses of word definitions whose execution has not been completed by the address interpreter. As a word definition passes control to another definition, the return point is placed on the return stack.

The return stack may cautiously be used for other values, such as loop control parameters, and for pointers for interpretation of text.

string

A sequence of 8-bit bytes containing ASCII characters, located in memory by an initial byte address and byte count.

transportability

This term indicates that equivalent execution results when a program is executed on other than the system on which it was created. See 'equivalent execution'.

true

A non-zero value represents the true condition flag. Any non-zero value will be accepted by a standard word as 'true'; all standard words return one when leaving a 'true' flag.

user area

An area in memory which contains the storage for user variables.

variables, user

So that the words of the FORTH vocabulary may be re-entrant (to different users), a copy of each system variable is maintained in the user area.

vocabulary

An ordered list of word definitions. Vocabulary lists are an advantage in reducing dictionary search time and in separating different word definitions that may carry the same name.

word

A sequence of characters terminated by at least one blank (or 'return'). Words are usually obtained via the input stream, from a terminal or mass storage device.

word definition

A named FORTH execution procedure compiled into the dictionary. Its execution may be defined in terms of machine code, as a sequence of compilation addresses or other compiled words. If named, it may be located by specifying this name and the vocabulary in which it is located.

word name

The name of a word definition. Standard names must be distinguished by their length and first thirty-one characters, and may not contain an ASCII null, blank, or 'return'.

word set

A group of FORTH word definitions listed by common characteristics. The standard word sets consist of:

Required Word Set
Nucleus Words
Interpreter Words
Compiler Words
Device Words

Extension Word Sets
32-bit Word Set
Assembler Word Set

Included as reference material only:
Reference Word Set

word set, compiler

Words which add new procedures to the dictionary or aid compilation by adding compilation addresses or data structures to the dictionary.

word set, devices

Words which allow access to mass storage and computer peripheral devices.

word set, interpreter

Words which support interpretation of text input from a terminal or mass storage by execution of corresponding dictionary entries, vocabularies, and terminal output.

word set, nucleus

The FORTH words generally defined in machine code that create the stacks and fundamental stack operators (virtual FORTH machine).

word set, reference

This set of words is provided as a reference document only, as a set of formerly standardized words and candidate words for standardization.

word set, required

The minimum words needed to compile and execute all Standard Programs.

word, standard

A named FORTH procedure definition, formally reviewed and accepted by the Standards Team. A serial number identifier {100..999} indicates a Standard Word. A functional alteration of a Standard Word will require assignment of a new serial number identifier.

The serial number identifier has no required use, other than to correlate the definition name with its unique Standard definition.

5. REFERENCES

The following documents are considered to be a portion of this Standard:

American Standard Code for Information Interchange,
American National Standards Institute, X3.4-1968

Webster's Collegiate Dictionary shall be used to resolve conflicts
in spelling and English word usage.

The following documents are noted as pertinent to the FORTH-79 Standard,
but are not part of this Standard.

FORTH-77, FORTH Users Group, FST-780314

FORTH-78, FORTH International Standards Team

FORTH-79, Reference Word Set

FORTH-79, Experimental Proposals

6. REQUIREMENTS

6.1 Documentation Requirements

Each Standard System and Standard Program shall be accompanied by a statement of the minimum (byte) requirements for:

1. System dictionary space
2. Application dictionary space
3. Data stack
4. Return stack
5. Mass storage contiguous block quantity required
6. An operator's terminal.

Each Standard System shall be provided with a statement of the system action upon each of the error conditions as identified in this Standard.

6.2 Testing Requirements

The following host computer configuration is specified as a minimum environment for testing against this Standard. Applications may require different capacities.

1. 2000 bytes of memory for application dictionary
2. Data stack of 64 bytes
3. Return stack of 48 bytes
4. Mass storage capacity of 32 blocks, numbered 0 through 31
5. One ASCII input/output device acting as an operator's terminal.

7. COMPLIANCE AND LABELING

The FORTH Standards Team hereby specifies the requirements for labeling of systems and applications so that the conditions for program portability may be established.

A system may use the specified labeling if it complies with the terms of this Standard, and meets the particular Word Set definitions.

A Standard Program (application) may use the specified labeling if it utilizes the specified standard system according to this Standard, and executes equivalently on any such system.

FORTH Standard

A system may be labeled 'FORTH-79 Standard' if it includes all of the Required Word Set in either source or object form, and complies with the text of this Standard. After executing "79-STANDARD" the dictionary must contain all of the Required Word Set in the vocabulary FORTH, as specified in this Standard.

Standard Sub-set

A system may be labeled 'FORTH-79 Standard Sub-set' if it includes a portion of the Required Word Set, and complies with the remaining text of this standard. However, no Required Word may be present with a non-standard definition.

Standard with Extensions

A system may be labeled 'FORTH-79 Standard with <name> Standard Extension(s)' if it comprises a FORTH-79 Standard System and one or more Standard Extension Word Set(s). The designation would be in the form:

'FORTH-79 Standard with Double-Number Standard Extensions'

8. USE

A FORTH Standard program may reference only the definitions of the Required Word Set, and definitions which are subsequently defined in terms of these words. Furthermore, a FORTH Standard program must use the standard words as required by any conventions of this Standard. Equivalent execution must result from Standard programs.

The FORTH system may share the dictionary space with the user's application, and the native addressing protocol of the host computer is beyond the scope of this Standard.

Therefore, in a Standard program, the user may only operate on data which was stored by the application. No exceptions!

A Standard Program may address:

1. parameter fields of variables, constants and DOES> words. A DOES> word's parameter field may only be addressed with respect to the address left by DOES> , itself.
2. dictionary space ALLOTEd.
3. data in mass storage block buffers. (Note restriction in BLOCK on latest buffer addressing.)
4. the user area and PAD.

A Standard Program may NOT address:

1. directly into the data or return stacks.
2. into a definition's name field, link field, or code field.
3. into a definition's parameter field if not stored by the application.

Further usage requirements are expected to be added for transporting programs between standard systems.

FORTH Standard definitions have a serial number assigned, in the range 100 thru 999. Neither a Standard System nor Standard Program may redefine these word names, within the FORTH vocabulary.

9. GLOSSARY NOTATION

Order

The Glossary definitions are listed in ASCII alphabetical order.

Stack Notation

The first line of each entry describes the execution of the definition:

```
stack parameters before execution
--- showing point of execution
stack parameters after execution

i.e., before --- after
```

In this notation, the top of the stack is to the right. Words may also be shown in context, when appropriate.

Attributes

Capitalized symbols indicate attributes of the defined words:

- C The word may only be used within a colon-definition.
- I Indicates that the word is IMMEDIATE and will execute during compilation, unless special action is taken.
- U A user variable.

Capitalization

Word names as used within the dictionary are conventionally written in upper case characters. Within this Standard lower case will be used when reference is made to the run-time machine code, not directly accessible, i.e., VARIABLE is the user word to create a variable. Each use of that variable makes use of a code sequence 'variable' which executes the function of the particular variable.

Pronunciation

The natural language pronunciation of FORTH names is given in double quotes (").

Stack Parameters

Unless otherwise stated, all references to numbers apply to 16-bit signed integers.

The implied range of values is shown as {from..to}. The content of an address is shown by double curly brackets, particularly for the contents of variables. i.e., BASE {{2..70}}

addr {0..65,535}

A value representing the address of a byte, within the FORTH standard memory space. This addressed byte may represent the first byte of a larger data field in memory.

byte {0..255}

A value representing an 8 bit byte. When in a larger field, the higher bits are zero.

char {0..127}

A value representing a 7 bit ASCII character code. When in a larger field, the higher bits are zero.

d {-2,147,483,648..2,147,483,647}

32 bit signed 'double' number. The most significant 16-bits, with sign, is most accessible on the stack.

flag

A numerical value with two logical states; 0= false, non-zero = true.

n {-32,768..32,767}

16 bit signed integer number.

Any other symbol refers to an arbitrary signed 16-bit integer in the range {-32,768..32,767}, unless otherwise noted.

Input Text

<name>

An arbitrary FORTH word accepted from the input stream. This notation refers to text from the input stream, not to values on the data stack. If the input stream is exhausted before encountering <name>, an error condition exists.

10. REQUIRED WORD SET

The words of the Required Word Set are grouped to show like characteristics. No implementation requirements should be inferred from this grouping.

Nucleus Words

```
! * */ */MOD + +! +loop - /
/MOD 0< 0= 0> 1+ 1- 2+ 2- <
= > >R ?DUP @ ABS AND begin C!
C@ colon CMOVE constant create D+
D< DEPTH DNEGATE do does>
DROP DUP else EXECUTE EXIT FILL I
if J LEAVE literal loop MAX MIN
MOD MOVE NEGATE NOT OR OVER PICK
R> R@ repeat ROLL ROT semicolon
SWAP then U* U/ U< until variable
while XOR
```

(note that the lower case entries refer to just the run-time code corresponding to a compiling word.)

Interpreter Words

```
# #> #S ' ( -TRAILING .
79-STANDARD <# >IN ? ABORT BASE BLK
CONTEXT CONVERT COUNT CR CURRENT
DECIMAL EMIT EXPECT FIND FORTH HERE
HOLD KEY PAD QUERY QUIT SIGN SPACE
SPACES TYPE U. WORD
```

Compiler Words

```
+LOOP , ." : ; ALLOT BEGIN
COMPILE CONSTANT CREATE DEFINITIONS DO
DOES> ELSE FORGET IF IMMEDIATE
LITERAL LOOP REPEAT STATE THEN UNTIL
VARIABLE VOCABULARY WHILE [ [COMPILE] ]
```

Device Words

```
BLOCK BUFFER EMPTY-BUFFERS LIST
LOAD SAVE-BUFFERS SCR UPDATE
```

!	n addr —	112
	Store n at address. "store"	
#	ud1 — ud2	158
	Generate from an unsigned double-number d1, the next ASCII character which is placed in an output string. Result d2 is the quotient after division by BASE and is maintained for further processing. Used between <# and #>. "sharp"	
#>	d — addr n	190
	End pictured numeric output conversion. Drop d, leaving the text address, and character count, suitable for TYPE. "sharp-greater"	
#S	ud — 0 0	209
	Convert all digits of an unsigned 32-bit number ud, adding each to the pictured numeric output text, until remainder is zero. A single zero is added to the output string if the number was initially zero. Use only between <# and #>. "sharp-s"	
'	— addr	I,171
	Used in the form:	
	' <name>	
	If executing, leave the parameter field address of the next word accepted from the input stream. If compiling, compile this address as a literal; later execution will place this value on the stack. An error condition exists if not found after a search of the CONTEXT and FORTH vocabularies. Within a colon-definition ' <name> is identical to [' <name>] LITERAL. "tick"	
(I,122
	Used in the form:	
	(ccc)	
	Accept and ignore comment characters from the input stream, until the next right parenthesis. As a word, the left parenthesis must be followed by one blank. It may be freely used while executing or compiling. An error condition exists if the input stream is exhausted before the right parenthesis. "paren" The right parenthesis is pronounced "close-paren"	
*	n1 n2 — n3	138
	Leave the arithmetic product of n1 times n2. "times"	

*	n1 n2 --- n3	138
	Leave the arithmetic product of n1 times n2. "times"	
*/	n1 n2 n3 --- n4	220
	Multiply n1 by n2 , divide the result by n3 and leave the quotient n4. n4 is rounded toward zero. The product of n1 times n2 is maintained as an intermediate 32-bit value for greater precision than the otherwise equivalent sequence: n1 n2 * n3 / "times-divide"	
*/MOD	n1 n2 n3 --- n4 n5	192
	Multiply n1 by n2, divide the result by n3 and leave the remainder n4 and quotient n5. A 32-bit intermediate product is used as for */. The remainder has the same sign as n1. "times-divide-mod"	
+	n1 n2 --- n3	121
	Leave the arithmetic sum of n1 plus n2. "plus"	
+!	n addr ---	157
	Add n to the 16-bit value at the address, by the convention given for +. "plus-store"	
+LOOP	n ---	I,C,141
	Add the signed increment n to the loop index using the convention for +, and compare the total to the limit. Return execution to the corresponding DO until the new index is equal to or greater than the limit (n>0), or until the new index is less than the limit (n<0). Upon the exiting from the loop, discard the loop control parameters, continuing execution ahead. Index and limit are signed integers in the range {-32,768..32,767}. "plus-loop"	
	(Comment: It is a historical precedent that the limit for n<0 is irregular. Further consideration of the characteristic is likely.)	
,	n ---	143
	Allot two bytes in the dictionary, storing n there. "comma"	
-	n1 n2 --- n3	134
	Subtract n2 from n1 and leave the difference n3. "minus"	
-TRAILING	addr n1 --- addr n2	148
	Adjust the character count n1 of a text string beginning at addr to exclude trailing blanks, i.e., the characters at addr+n2 to addr+n1-1 are blanks. An error condition exists if n1 is negative. "dash-trailing"	

- `n` --- 193
Display `n` converted according to BASE in a free-field format with one trailing blank. Display only a negative sign. "dot"
- `."` I,133
Interpreted or used in a colon-definition in the form:
`." cccc"`
Accept the following text from the input stream, terminated by " (double-quote). If executing, transmit this text to the selected output device. If compiling, compile so that later execution will transmit the text to the selected output device. At least 127 characters are allowed in the text. If the input stream is exhausted before the terminating double-quote, an error condition exists. "dot-quote"
- `/` `n1 n2 --- n3` 178
Divide `n1` by `n2` and leave the quotient `n3`. `n3` is rounded toward zero. "divide"
- `/MOD` `n1 n2 --- n3 n4` 198
Divide `n1` by `n2` and leave the remainder `n3` and quotient `n4`. `n3` has the same sign as `n1`. "divide-mod"
- `0<` `n --- flag` 144
True if `n` is less than zero (negative). "zero-less"
- `0=` `n --- flag` 180
True if `n` is zero. "zero-equals"
- `0>` `n --- flag` 118
True if `n` is greater than zero. "zero-greater"
- `1+` `n --- n+1` 107
Increment `n` by one, according to the operation for `+`. "one-plus"
- `1-` `n --- n-1` 105
Decrement `n` by one, according to the operation for `-`. "one-minus"
- `2+` `n --- n+2` 135
Increment `n` by two, according to the operation for `+`. "two-plus"

- 2- n --- n-2 129
- Decrement n by two, according to the operation for -. "two-minus"
- 79-STANDARD 119
- Execute assuring that a FORTH-79 Standard system is available, otherwise an error condition exists.
- :
- 116
- A defining word used in the form:
- : <name> . . . ;
- Select the CONTEXT vocabulary to be identical to CURRENT. Create a dictionary entry for <name> in CURRENT, and set compile mode. Words thus defined are called 'colon-definitions'. The compilation addresses of subsequent words from the input stream which are not immediate words are stored into the dictionary to be executed when <name> is later executed. IMMEDIATE words are executed as encountered.
- If a word is not found after a search of the CONTEXT and FORTH vocabularies, conversion and compilation of a literal number is attempted, with regard to the current BASE; that failing, an error condition exists. "colon"
- ;
- I,C,196
- Terminate a colon-definition and stop compilation. If compiling from mass storage and the input stream is exhausted before encountering ; an error condition exists. "semi-colon"
- < n1 n2 --- flag 139
- True if n1 is less than n2.
- 32768 32767 < must return true.
- 32768 0 must be distinguished. "less-than"
- <# 169
- Initialize pictured numeric output. The words:
- <# # #S HOLD SIGN #>
- can be used to specify the conversion of a double-precision number into an ASCII character string stored in right-to-left order. "less-sharp"
- = n1 n2 --- flag 173
- True if n1 is equal to n2. "equals"

>	n1 n2 --- flag	102
	True if n1 is greater than n2. "greater-than"	
>IN	--- addr	U,201
	Leave the address of a variable which contains the present character offset within the input stream {{0..1023}} "to-in"	
	See: WORD (." FIND	
>R	n ---	C,200
	Transfer n to the return stack. Every >R must be balanced by a R> in the same control structure nesting level of a colon-definition. "to-r"	
?	addr ---	194
	Display the number at address, using the format of ".". "question-mark"	
?DUP	n --- n (n)	184
	Duplicate n if it is non-zero. "query-dup"	
@	addr --- n	199
	Leave on the stack the number contained at addr. "fetch"	
ABORT		101
	Clear the data and return stacks, setting execution mode. Return control to the terminal.	
ABS	n1 --- n2	108
	Leave the absolute value of a number. "absolute"	
ALLOT	n ---	154
	Add n bytes to the parameter field of the most recently defined word.	
AND	n1 n2 --- n3	183
	Leave the bitwise logical 'and' of n1 and n2.	
BASE	--- addr	U,115
	Leave the address of a variable containing the current input-output numeric conversion base. {{2..70}}	

BEGIN

I,C,147

Used in a colon-definition in the forms:

```
BEGIN . . . flag UNTIL          or
BEGIN . . . flag WHILE ... REPEAT
```

BEGIN marks the start of a word sequence for repetitive execution. A BEGIN-UNTIL loop will be repeated until flag is true. A BEGIN-WHILE-REPEAT loop will be repeated until flag is false. The words after UNTIL or REPEAT will be executed when either loop is finished. flag is always dropped after being tested.

BLK

--- addr

U,132

Leave the address of a variable containing the number of the mass storage block being interpreted as the input stream.

If the content is zero, the input stream is taken from the terminal. "b-l-k" {{unsigned-number}}

BLOCK

n --- addr

191

Leave the address of the first byte in block n. If the block is not already in memory, it is transferred from mass storage into whichever memory buffer has been least recently accessed. If the block occupying that buffer has been UPDATED (i.e. modified), it is rewritten onto mass storage before block n is read into the buffer. n is an unsigned number. If correct mass storage read or write is not possible, an error condition exists. Only data within the latest block referenced by BLOCK is valid by byte address, due to sharing of the block buffers.

BUFFER

n --- addr

130

Obtain the next block buffer, assigning it to block n. The block is not read from mass storage. If the previous contents of the buffer has been marked as UPDATED, it is written to mass storage. If correct writing to mass storage is not possible, an error condition exists. The address left is the first byte within the buffer for data storage. n is an unsigned number.

C!

n addr ---

219

Store the least significant 8-bits of n at addr. "c-store"

C@

addr --- byte

156

Leave on the stack the contents of the byte at addr (with higher bits zero, in a 16-bit field). "c-fetch"

CMOVE

addr1 addr2 n ---

153

Move n bytes beginning at address addr1 to addr2. The contents of addr1 is moved first proceeding toward high memory. If n is zero or negative nothing is moved. "c-move"

COMPILE

C,146

When a word containing COMPILE executes, the 16-bit value following the compilation address of COMPILE is copied (compiled) into the dictionary. i.e., COMPILE DUP will copy the compilation address of DUP.

COMPILE [0 ,] will copy zero.

CONSTANT

n ---

185

A defining word used in the form:

n CONSTANT <name>

to create a dictionary entry for <name>, leaving n in its parameter field. When <name> is later executed, n will be left on the stack.

CONTEXT

— addr

U,151

Leave the address of a variable specifying the vocabulary in which dictionary searches are to made, during interpretation of the input stream.

CONVERT

d1 addr1 — d2 addr2

195

Convert to the equivalent stack number the text beginning at addr1+1 with regard to BASE. The new value is accumulated into double number d1, being left as d2. addr2 is the address of the first non-convertible character.

COUNT

addr — addr+1 n

159

Leave the address addr+1 and the character count of text beginning at addr. The first byte at addr must contain the character count n. Range of n is {0..255}.

CR

160

Cause a carriage-return and line-feed to occur at the current output device. "c-r"

CREATE

239

A defining word used in the form:

CREATE <name>

to create a dictionary entry for <name>, without allocating any parameter field memory. When <name> is subsequently executed, the address of the first byte of <name>'s parameter field is left on the stack.

CURRENT

— addr

U,137

Leave the address of a variable specifying the vocabulary into which new word definitions are to be entered.

- D+ d1 d2 — d3 241
 Leave the arithmetic sum of d1 plus d2. "d-plus"
- D< d1 d2 — flag 244
 True if d1 is less than d2. "d-less-than"
- DECIMAL 197
 Set the input-output numeric conversion base to ten.
- DEFINITIONS 155
 Set CURRENT to the CONTEXT vocabulary so that subsequent definitions will be created in the vocabulary previously selected as CONTEXT.
- DEPTH — n 238
 Leave the number of the quantity of 16-bit values contained in the data stack, before n was added.
- DNEGATE d — -d 245
 Leave the two's complement of a double number.
- DO n1 n2 — I,C,142
 Use in a colon-definition:
 DO . . . LOOP or
 DO . . . +LOOP
 Begin a loop which will terminate based on control parameters. The loop index begins at n2, and terminates based on the limit n1. At LOOP or +LOOP, the index is modified by a positive or negative value. The range of a DO-LOOP is determined by the terminating word.
 DO-LOOP may be nested. Capacity for three levels of nesting is specified as a minimum for standard systems.
- DOES> I,C,168
 Define the run-time action of a word created by a high-level defining word. Used in the form:
 : <name> . . . CREATE . . . DOES> . . . ;
 and then <name> <namex>
 Marks the termination of the defining part of the defining word <name> and begins the definition of the run time action for words that will later be defined by <name>. On execution of <namex> the sequence of words between DOES> and ; will be executed, with the address of <namex>'s parameter field on the stack. "does"

DROP	n ---	233
Drop the top number from the stack.		
DUP	n --- n n	205
Leave a copy of the top stack number.		
ELSE		I,C,167
Used in a colon-definition in the form:		
IF . . . ELSE . . . THEN		
ELSE executes after the true part following IF. ELSE forces execution to skip till just after THEN. It has no effect on the stack. (See IF)		
EMIT	char ---	207
Transmit character to the current output device.		
EMPTY-BUFFERS		145
Mark all block buffers as empty, without necessarily affecting their actual contents. UPDATED blocks are not written to mass storage.		
EXECUTE	addr ---	163
Execute the dictionary entry whose compilation address is on the stack.		
EXIT		C,117
When compiled within a colon-definition, terminate execution of that definition, at that point. May not be used within a DO...LOOP.		
EXPECT	addr n ---	189
Transfer characters from the terminal beginning at addr, upward, until a "return" or the count of n has been received. Take no action for n less than or equal to zero. One or two nulls are added at the end of text.		
FILL	addr n byte ---	234
Fill memory beginning at address with a sequence of n copies of byte. If the quantity n is less than or equal to zero, take no action.		
FIND	--- addr	203
Leave the compilation address of the next word name, which is accepted from the input stream. If that word cannot be found in the dictionary after a search of CONTEXT and FORTH leave zero.		

FORGET

186

Execute in the form:

FORGET <name>

Delete from the dictionary <name> (which is in the CURRENT vocabulary) and all words added to the dictionary after <name>, regardless of their vocabulary. Failure to find <name> in CURRENT or FORTH is an error condition.

FORTH

I,187

The name of the primary vocabulary. Execution makes FORTH the CONTEXT vocabulary.

New definitions become a part of FORTH until a differing CURRENT vocabulary is established.

User vocabularies conclude by 'chaining' to FORTH, so it should be considered that FORTH is 'contained' within each user's vocabulary.

HERE --- addr

188

Return the address of the next available dictionary location.

HOLD char ---

175

Insert char into a pictured numeric output string. May only be used between <# and #>.

I --- n

C,136

Copy the loop index onto the data stack. May be only used in the form:

DO . . . I . . . LOOP or DO . . . I . . . +LOOP

IF flag ---

I,C,210

Used in a colon-definition in the forms:

flag IF . . . ELSE . . . THEN or
flag IF . . . THEN

If flag is true, the words following IF are executed and the words following ELSE are skipped. The ELSE part is optional.

If flag is false, words between IF and ELSE, or between IF and THEN (when no ELSE is used), are skipped. IF-ELSE-THEN conditionals may be nested.

IMMEDIATE

103

Mark the most recently made dictionary entry as a word which will be executed when encountered during compilation rather than compiled.

- J** --- n C,225
- Return the index of the next outer loop. May be used only within a nested DO-LOOP in the form:
- DO . . . DO . . . J . . . LOOP . . . LOOP
- KEY** --- char 100
- Leave the ASCII value of the next available character from the current input device.
- LEAVE** C,213
- Force termination of a DO-LOOP at the next LOOP or +LOOP by setting the loop limit equal to the current value of the index. The index itself remains unchanged, and execution proceeds normally until the loop terminating word is encountered.
- LIST** n --- 109
- List the ASCII symbolic contents of screen n on the current output device, setting SCR to contain n. n is unsigned.
- LITERAL** n --- I,215
- If compiling, then compile the stack value n as a 16-bit literal, which when later executed, will leave n on the stack.
- LOAD** n --- 202
- Begin interpretation of screen n by making it the input stream; preserve the locators of the present input stream (from >IN and BLK). If interpretation is not terminated explicitly it will be terminated when the input stream is exhausted. Control then returns to the input stream containing LOAD, determined by the input stream locators >IN and BLK.
- LOOP** I,C,124
- Increment the DO-LOOP index by one, terminating the loop if the new index is equal to or greater than the limit. The limit and index are signed numbers in the range {-32,768..32,767}.
- MAX** n1 n2 --- n3 218
- Leave the greater of two numbers. "max"
- MIN** n1 n2 --- n3 127
- Leave the lesser of two numbers. "min"

MOD	n1 n2 --- n3	104
	Divide n1 by n2, leaving the remainder n3, with the same sign as n1. "mod"	
MOVE	addr1 addr2 n ---	113
	Move the specified quantity n of 16-bit memory cells beginning at addr1 into memory at addr2. The contents of addr1 is moved first. If n is negative or zero, nothing is moved.	
NEGATE	n --- -n	177
	Leave the two's complement of a number, i.e., the difference of 0 less n.	
NOT	flag1 --- flag2	165
	Reverse the boolean value of flag1. This is identical to 0=.	
OR	n1 n2 --- n3	223
	Leave the bitwise inclusive-or of two numbers.	
OVER	n1 n2 --- n1 n2 n1	170
	Leave a copy of the second number on the stack.	
PAD	--- addr	226
	The address of a scratch area used to hold character strings for intermediate processing. The minimum capacity of PAD is 64 characters (addr through addr+63).	
PICK	n1 --- n2	240
	Return the contents of the n1-th stack value, not counting n1 itself. An error condition results for n less than one.	
	2 PICK is equivalent to OVER. {1 .. n}	
QUERY		235
	Accept input of up to 80 characters (or until a 'return') from the operator's terminal, into the terminal input buffer. WORD may be used to accept text from this buffer as the input stream, by setting >IN and BLK to zero.	
QUIT		211
	Clear the return stack, setting execution mode, and return control to the terminal. No message is given.	

R> --- n C,110
 Transfer n from the return stack to the data stack. "r-from"

R@ --- n C,228
 Copy the number on the top of the return stack to the data stack.
 "r-fetch"

REPEAT I,C,120
 Used in a colon-definition in the form:
 BEGIN ... WHILE ... REPEAT
 At run-time, REPEAT returns to just after the corresponding BEGIN.

ROLL n --- 236
 Extract the n-th stack value to the top of the stack, not counting n
 itself, moving the remaining values into the vacated position. An error
 condition results for n less than one. {1 .. n}
 3 ROLL = ROT
 1 ROLL = null operation

ROT n1 n2 n3 --- n2 n3 n1 212
 Rotate the top three values, bringing the deepest to the top. "rote"

SAVE-BUFFERS 221
 Write all blocks to mass-storage that have been flagged as UPDATED. An
 error condition results if mass-storage writing is not completed.

SCR --- addr U,217
 Leave the address of a variable containing the number of the screen most
 recently listed. "s-c-r" unsigned-number

SIGN n --- C,140
 Insert the ASCII "-" (minus sign) into the pictured numeric output string,
 if n is negative.

SPACE 232
 Transmit an ASCII blank to the current output device.

SPACES	n —	231
Transmit n spaces to the current output device. Take no action for n of zero or less.		
STATE	— addr	U,164
Leave the address of the variable containing the compilation state. A non-zero content indicates compilation is occurring, but the value itself may be installation dependent.		
SWAP	n1 n2 — n2 n1	230
Exchange the top two stack values.		
THEN		I,C,161
Used in a colon-definition, in the form:		
<pre> IF . . . ELSE . . . THEN or IF . . . THEN </pre>		
THEN is the point where execution resumes after ELSE or IF (when no ELSE is present).		
TYPE	addr n —	222
Transmit n characters beginning at address to the current output device. No action takes place for n less than or equal to zero.		
U*	un1 un2 — ud3	242
Perform an unsigned multiplication of un1 by un2, leaving the double number product ud3. All values are unsigned. "u-times"		
U.	un —	106
Display un converted according to BASE as an unsigned number, in a free-field format, with one trailing blank. "u-dot"		
U/MOD	ud1 un2 — un3 un4	243
Perform the unsigned division of double number ud1 by un2, leaving the remainder un3, and quotient un4. All values are unsigned. "u-divide-mod"		
U<	un1 un2 — flag	150
Leave the flag representing the magnitude comparison of un1 < un2 where un1 and un2 are treated as 16-bit unsigned integers. "u-less-than"		

UNTIL flag --- I,C,237

Within a colon-definition, mark the end of a BEGIN-UNTIL loop, which will terminate based on a flag. If flag is true, the loop is terminated. If flag is false, execution returns to the first word after BEGIN. BEGIN-UNTIL structures may be nested.

UPDATE 229

Mark the most recently referenced block as modified. The block will subsequently be automatically transferred to mass storage should its memory buffer be needed for storage of a different block, or upon execution of SAVE-BUFFERS.

VARIABLE 227

A defining word executed in the form:

VARIABLE <name>

to create a dictionary entry for <name> and allot two bytes for storage in the parameter field. The application must initialize the stored value. When <name> is later executed, it will place the storage address on the stack.

VOCABULARY 208

A defining word executed in the form:

VOCABULARY <name>

to create (in the CURRENT vocabulary) a dictionary entry for <name>, which specifies a new ordered list of word definitions. Subsequent execution of <name> will make it the CONTEXT vocabulary. When <name> becomes the CURRENT vocabulary (see DEFINITIONS), new definitions will be created in that list.

In lieu of any further specification, new vocabularies 'chain' to FORTH. That is, when a dictionary search through a vocabulary is exhausted, FORTH will be searched.

WHILE flag — I,C,149

Used in a colon-definition in the form:

BEGIN ... flag WHILE ... REPEAT

Select conditional execution based on the flag. On a true flag, continue execution through to REPEAT, which then returns back to just after BEGIN. On a false flag, skip execution to just after REPEAT, exiting the structure.

WORD char --- addr 181

Receive characters from the input stream until the non-zero delimiting character is encountered or the input stream is exhausted, ignoring leading delimiters. The characters are stored as a packed string with the character count in the first character position. The actual delimiter encountered (char or null) is stored at the end of the text but not included in the count. If the input stream was exhausted as WORD is called, then a zero length will result. The address of the beginning of this packed string is left on the stack.

XOR n1 n2 --- n3 174

Leave the bitwise exclusive-or of two numbers. "x-or"

[I,125

End the compilation mode. The text from the input stream is subsequently executed. See] "left-bracket"

[COMPILE] I,C,179

Used in a colon-definition in the form:

[COMPILE] <name>

Force compilation of the following word. This allows compilation of an IMMEDIATE word when it would otherwise be executed. "bracket-compile"

] 126

Set the compilation mode. The text from the input stream is subsequently compiled. See ["right-bracket"

11. EXTENSION WORD SETS

11.1 DOUBLE NUMBER WORD SET

2! d addr —

Store d in 4 consecutive bytes beginning at addr, as for a double number.
"two-store"

2@ addr — d

Leave on the stack the contents of the four consecutive bytes beginning at
addr, as for a double number. "two-fetch"

2CONSTANT d ---

A defining word used in the form:

 d 2CONSTANT <name>

to create a dictionary entry for <name>, leaving d in its parameter
field. When <name> is later executed, d will be left on the stack.
"two-constant"

2DROP d ---

Drop the top double number on the stack. "two-drop"

2DUP d — d d

Duplicate the top double number on the stack. "two-dup"

2OVER d1 d2 — d1 d2 d1

Leave a copy of the second double number on the stack. "two-over"

2ROT d1 d2 d3 — d2 d3 d1

Rotate the third double number to the top of the stack. "two-rote"

2SWAP d1 d2 — d2 d1

Exchange the top two double numbers on the stack. "two-swap"

2VARIABLE

A defining word used in the form:

 2VARIABLE <name>

to create a dictionary entry of <name> and assign 4 bytes for storage in
the parameter field. When <name> is later executed, it will leave the
address of the first byte of its parameter field on the stack. "two-
variable"

- D+ d1 d2 --- d3 241
 Leave the arithmetic sum of d1 and d2. "d-plus"
- D- d1 d2 --- d3
 Subtract d2 from d1 and leave the difference d3. "d-minus"
- D. d --- 129
 Display d converted according to BASE in a free-field format, with one trailing blank. Display the sign only if negative. "d-dot"
- D.R d n ---
 Display d converted according to BASE, right aligned in an n character field. Display the sign only if negative. "d-dot-r"
- D0= d --- flag
 Leave true if d is zero. "d-zero-equals"
- D< d1 d2 --- flag 244
 True if d1 is less than d2. "d-less"
- D= d1 d2 --- flag
 True if d1 equals d2. "d-equal"
- DABS d1 --- d2
 Leave as a positive double number d2, the absolute value of a double number, d1. {0..2,147,483,647} "d-abs"
- DMAX d1 d2 --- d3
 Leave the larger of two double numbers. "d-max"
- DMIN d1 d2 --- d3
 Leave the smaller of two double numbers. "d-min"
- DNEGATE d --- -d 245
 Leave the double number two's complement of a double number, i.e., the difference 0 less d. "d-negate"
- DU< ud1 ud2 --- flag
 True if ud1 is less than ud2. Both numbers are unsigned. "d-u-less"

11.2 ASSEMBLER WORD SET

;CODE

C,I,206

Used in the form:

: <name> . . . ;CODE

Stop compilation and terminate a defining word <name>. ASSEMBLER becomes the CONTEXT vocabulary. When <name> is executed in the form:

<name> <namex>

to define the new <namex>, the execution address of <namex> will contain the address of the code sequence following the ;CODE in <name>. Execution of any <namex> will cause this machine code sequence to be executed. "semi-colon-code"

ASSEMBLER

I,166

Select assembler as the CONTEXT vocabulary.

CODE

111

A defining word used in the form:

CODE <name> . . . END-CODE

to create a dictionary entry for <name> to be defined by a following sequence of assembly language words. ASSEMBLER becomes the context vocabulary.

END-CODE

Terminate a code definition, resetting the CONTEXT vocabulary to the CURRENT vocabulary. If no errors have occurred, the code definition is made available for use.

12. EXPERIMENTAL PROPOSALS

No Experimental Proposals were submitted for publication.

REFERENCE WORD SET

This word set is furnished as a reference document. It is a set of formerly standardized words and candidate words for standardization.

REFERENCE WORD SET
FORTH-79

The Reference Word Set contains both Standard Word Definitions (with Serial number identifiers in the range 100 through 999), and uncontrolled word definitions.

Uncontrolled definitions are included for public reference of words that have present usage and/or are candidates for future standardization.

No restrictions are placed on the definition or usage of uncontrolled words. However, use of these names for procedures differing from the given definitions is discouraged.

```
!BITS      n1  addr  n2  —
```

Store the value of n1 masked by n2 into the equivalent masked part of the contents of addr, without affecting bits outside the mask. "store-bits"

$$** \quad n1 \quad n2 \quad \text{---} \quad n3$$

Leave the value of `n1` to the power `n2`. "power"

+BLOCK n1 — n2

Leave the sum of n1 plus the number of the block being interpreted. n1 and n2 are unsigned. "plus-block"

```

-      ( addr ) flag

```

Used in the form:

-' <name>

Leave the parameter field of <name> beneath zero (false) if the name can be found in the CONTEXT vocabulary; leave only true if not found. "dash-tick".

→ I, 131

Continue interpretation on the next sequential block. May be used within a colon-definition that crosses a block boundary. "next-block"

```
-MATCH      addr1 n1  addr2 n2
            --- addr3 f
```

```
Attempt to find the n2-character string beginning at addr2 somewhere in
the n1-character string beginning at addr1. Return the last +1 character
address addr3 of the match point and a flag which is zero if a match
exists. "dash-match"
```

-TEXT addr1 n1 addr2 --- n2

Compare two strings over the length n1 beginning at addr1 and addr2. Return zero if the strings are equal. If unequal, return n2, the difference between the last characters compared:

addr1(i) - addr2(i)

"dash-text"

.R n1 n2 ---

Print n1 right aligned in a field of n2 characters, according to BASE. If n2 is less than 1, no leading blanks are supplied. "dot-r"

/LOOP n ---

A DO-LOOP terminating word. The loop index is incremented by the unsigned magnitude of n. Until the resultant index exceeds the limit, execution returns to just after the corresponding DO; otherwise, the index and limit are discarded. Magnitude logic is used. "up-loop"

1+! addr ---

Add one to the 16-bit contents at addr. "one-plus-store"

1-! addr ---

Subtract 1 from the 16-bit contents at addr. "one-minus-store"

2* n1 --- n2

Leave 2*(n1). "two-times"

2/ n1 --- n2

Leave (n1)/2. "two-divide"

:: Used to specify a new defining word:

C

```
: <name> . . .
  ;: . . . ;
<name> <namex>
```

When <name> is executed, it creates an entry for the new word <namex>. Later execution of <namex> will execute the sequence of words between :: and ; , with the address of the first (if any) parameters associated with <namex> on the stack. "semi-colon-colon"

;S

Stop interpretation of a block. For execution only. "semi-S"

<> n1 n2 — flag

Leave true if n1 is not equal to n2. "not-equal"

<BUILDS

C

Used in conjunction with DOES> in defining words, in the form:

```
: <name> . . . <BUILDS . . .
  DOES> . . . ;
and then <name> <namex>
```

When <name> executes, <BUILDS creates a dictionary entry for the new <namex>. The sequence of words between <BUILDS and DOES> established a parameter field for <namex>. When <namex> is later executed, the sequence of words following DOES> will be executed, with the parameter field address of <namex> on the data stack. "builds"

<CMOVE addr1 addr2 n —

Copy n bytes beginning at addr1 to addr2. The move proceeds within the bytes from high memory toward low memory. "reverse-c-move"

>< n1 — n2

Swap the high and low bytes within n1. "byte-swap"

>MOVE< addr1 addr2 n —

Move n bytes beginning at addr1 to the memory beginning at addr2. During this move, the order of each byte pair is reversed. "byte-swap-move"

@BITS addr n1 — n2

Return the 16-bits at addr masked by n1. "fetch-bits"

ABORT" flag —

I,C

Used in a colon-definition in the form:

```
ABORT" stack empty"
```

If the flag is true, print the following text, till ". Then execute ABORT. "abort-quote"

AGAIN

I,C,114

Effect an unconditional jump back to the start of a BEGIN-AGAIN loop.

ASCII — char (executing)
 — (compiling)

I,C

Leave the ASCII character value of the next non-blank character in the input stream. If compiling, compile it as a literal, which will be later left when executed.

ALL ABOUT FORTH

I' -- n C

Used within a colon-definition executed only from within a DO-LOOP to return the corresponding loop index. "i-prime"

IFEND

Terminate a conditional interpretation sequence begun by IFTRUE.

IFTRUE flag ---

Begin an

IFTRUE ... OTHERWISE ... IFEND

conditional sequence. These conditional words operate like

IF ... ELSE ... THEN

except that they cannot be nested, and are to be used only during interpretation. In conjunction with the words [and] they may be used within a colon-definition to control compilation, although they are not to be compiled.

INDEX n1 n2 ---

Print the first line of each screen over the range {n1..n2}. This displays the first line of each screen of source text, which conventionally contains a title.

INTERPRET

Begin interpretation at the character indexed by the contents of >IN relative to the block number contained in BLK, continuing until the input stream is exhausted. If BLK contains zero, interpret characters from the terminal input buffer.

K -- n C

Within a nested DO-LOOP, return the index of the second outer loop.

LAST --- addr

A variable containing the address of the beginning of the last dictionary entry made, which may not yet be a complete or valid entry.

LINE n --- addr

Leave the address of the beginning of line n for the screen whose number is contained in SCR. The range of n is {0..15}.

LINELOAD n1 n2 ---

Begin interpretation at line n1 of screen n2.

- 6 -

LOADS n —

A defining word used in the form:

n LOADS <name>

When <name> is subsequently executed, block n will be loaded.

MAP0 — addr

A variable pointing to the first location in the tape map.

MASK n1 — n2

Leave a mask of n1 most significant bits if n1 is positive, or n least significant bits if n1 is negative.

MS n —

Delay for approximately n milliseconds.

NAND n1 n2 — n3

The one's complement of the logical and of n1 and n2.

NOR n1 n2 — n3

The one's complement of the logical or of n1 and n2.

NUMBER addr — n

Convert the count and character string at addr, to a signed 32-bit integer, using the current base. If numeric conversion is not possible, an error condition exists. The string may contain a preceding negative sign.

O. n —

Print n in octal format with one trailing blank. The value in BASE is unaffected. "o-dot"

OCTAL

Set the number conversion base to 8.

OFFSET — addr

128

A variable that contains the offset added to the block number on the stack by BLOCK to determine the actual physical block number. The user must add any desired offset when utilizing BUFFER.

OTHERWISE

An interpreter-level conditional word. See IFTRUE.

PAGE

Clear the terminal screen or perform an action suitable to the output device currently active.

READ-MAP

Read to the next file mark on tape constructing a correspondence table in memory (the map) relating physical block position to logical block number. The tape should normally be rewound to its load point before executing READ-MAP.

REMEMBER

A defining word used in the form:

REMEMBER <name>

Defines a word which, when executed, will cause <name> and all subsequently defined words to be deleted from the dictionary. <name> may be compiled into and executed from a colon definition. The sequence

DISCARD REMEMBER DISCARD

provides a standardized preface to any group of transient word definitions.

REWIND

Rewind the tape to its load point, setting CUR=1.

ROTATE n1 n2 — n3

Rotate the value n1 left n2 bits if n2 is positive, right n2 bits if n2 is negative. Bits shifted out of one end of the cell are shifted back in at the opposite end.

S0 — addr

Returns the address of the bottom of the stack, when empty. "s-zero"

SET n addr —

A defining word used in the form:

n addr SET <name>

Defines a word <name> which, when executed, will cause the value n to be stored at address.

SHIFT n1 n2 — n3

Logical shift n1 left n2 bits if n2 is positive, right if n2 is negative. Zeros are shifted into vacated bit positions.

- SP@** --- addr 214
- Return the address of the top of the stack, just before SP@ was executed.
"s-p-fetch"
- TEXT** c ---
- Accept characters from the input stream, as for WORD, into PAD, blank-filling the remainder of PAD to 64 characters.
- THRU** n1 n2 ---
- Load consecutively the blocks from n1 through n2.
- U.R** un1 n2 --- 216
- Output un1 as an unsigned number right justified in a field n2 characters wide. If n2 is smaller than the characters required for n1, no leading spaces are given. "u-dot-r"
- USER** n ---
- A defining word used in the form:
- n USER <name>
- which creates a user variable <name>. n is the cell offset within the user area where the value for <name> is stored. Execution of <name> leaves its absolute user area storage address.
- VLIST**
- List the word names of the CONTEXT vocabulary starting with the most recent definition.
- WHERE**
- Output information about the status of FORTH, (e.g., after an error abort). Indicate at least the last word compiled and the last block accessed.
- \.LOOP** n --- I,C
- A DO-LOOP terminating word. The loop index is decremented by n and the loop terminated when the resultant index becomes equal to or less than the limit. Magnitude logic is used, and n must be positive. "down-loop"

FORTH-83 STANDARD

A Publication of the FORTH Standards Team

August 1983

FORTH-83 STANDARD

A PUBLICATION OF THE FORTH STANDARDS TEAM

AUGUST 1983

FORTH-83 STANDARD

COPYRIGHT © 1983 FORTH STANDARDS TEAM

Permission is hereby granted to reproduce this document in whole or in part provided that such reproductions refer to the fact that the copied material is subject to copyright by the FORTH Standards Team. No changes or modifications may be made to the copied material unless it is clearly indicated that such changes were not incorporated in the original copyrighted work.

The existence of a FORTH Standard does not in any respect preclude anyone, whether the individual has approved this Standard or not, from implementing, marketing, purchasing or using products, processes, or procedures not conforming to the Standard. FORTH Standards are subject to periodic review and users are cautioned to obtain the latest editions.

ISBN 0-914699-03-2

**FORTH STANDARDS TEAM
P.O. BOX 4545
MOUNTAIN VIEW, CA 94040
USA**

FORTH-83 STANDARD

TABLE OF CONTENTS

1. FOREWORD	1
2. PURPOSE	2
3. SCOPE	2
4. TRADEOFFS	3
5. DEFINITIONS OF TERMS	4
6. REFERENCES	12
7. REQUIREMENTS	13
8. COMPLIANCE AND LABELING	15
9. USAGE	17
10. ERROR CONDITIONS	20
11. GLOSSARY NOTATION	22
12. REQUIRED WORD SET	25
13. DOUBLE NUMBER EXTENSION WORD SET	41
14. ASSEMBLER EXTENSION WORD SET	44
15. SYSTEM EXTENSION WORD SET	46
16. CONTROLLED REFERENCE WORDS	48
APPENDICES	
A. FORTH STANDARDS TEAM MEMBERSHIP	51
B. UNCONTROLLED REFERENCE WORDS	54
C. EXPERIMENTAL PROPOSALS	60
C.1 SEARCH ORDER SPECIFICATION AND CONTROL	61
C.2 DEFINITION FIELD ADDRESS CONVERSION OPERATORS	66
D. STANDARDS TEAM CHARTER	69
E. PROPOSAL/COMMENT FORM AND INSTRUCTIONS	78

FORTH-83 STANDARD

1. FOREWORD**1. FOREWORD**

FORTH is an integrated programming approach and computer language. FORTH was invented by Mr. Charles Moore specifically to increase programmer productivity in the development of computer related applications without sacrificing machine efficiency. FORTH is a layered environment containing the elements of a computer language as well as those of an operating system and a machine monitor. This extensible, layered environment provides for highly interactive program development and testing.

In the interests of transportability of application software written in FORTH, standardization efforts began in the mid-1970s by the European FORTH User's Group (EFUG). This effort resulted in the FORTH-77 Standard. As the language continued to evolve, an interim FORTH-78 Standard was published by the FORTH Standards Team. Following FORTH Standards Team meetings in 1979 the FORTH-79 Standard was published in 1980.

The FORTH Standards Team is comprised of individuals who have a great variety of experience and technical expertise with FORTH. The FORTH Standards Team consists of both users and implementers. Comments, proposals, and correspondence should be mailed to: FORTH Standards Team, P.O. Box 4545, Mountain View, CA 94040 USA.

FORTH's extensibility allows the language to be expanded and adapted to special needs and different hardware systems. A programmer or vendor may choose to strictly adhere with the standard, but the choice to deviate is acknowledged as beneficial and sometimes necessary. If the standard does not explicitly specify a requirement or restriction, a system or application may utilize any choice without sacrificing compliance to the standard provided that the system or application remains transportable and obeys the other requirements of the standard.

2. PURPOSE**2. PURPOSE**

The purpose of this standard is to allow transportability of FORTH-83 Standard Programs in source form among FORTH-83 Standard Systems. A standard program shall execute equivalently on all standard systems.

3. SCOPE

This standard shall apply to any FORTH-83 Standard Program executing on any FORTH-83 Standard System, provided sufficient computer resources (memory, mass storage) are available.

4. TRADEOFFS

4. TRADEOFFS

When conflicting choices are made, the following order guides the Standards Team:

- 1) Functional correctness - known bounds, non-ambiguous
- 2) Portability - repeatable results when programs are transported among Standard Systems;
- 3) Simplicity;
- 4) Naming clarity - uniformity of expression using descriptive rather than procedural names, i.e., [COMPILE] rather than 'C, and ALLOT rather than DP+! ;
- 5) Generality;
- 6) Execution speed;
- 7) Memory compactness;
- 8) Compilation speed;
- 9) Historical continuity;
- 10) Pronounceability;
- 11) Teachability.

5. DEFINITIONS OF TERMS

5. DEFINITIONS OF TERMS

These are the definitions of the terms used within this Standard.

address, byte

An unsigned 16-bit number that locates an 8-bit byte in a standard FORTH address space over the range {0..65,535}. It may be a native machine address or a representation on a virtual machine, locating the addr-th byte within the virtual byte address space. Addresses are treated as unsigned numbers. See: "arithmetic, two's complement"

address, compilation

The numerical value compiled for a FORTH word definition which identifies that definition. The address interpreter uses this value to locate the machine code corresponding to each definition.

address, native machine

The natural address representation of the host computer.

address, parameter field

The address of the first byte of memory associated with a word definition for the storage of compilation addresses (in a colon definition), numeric data, text characters, etc.

arithmetic, two's complement

Arithmetic is performed using two's complement integers within a field of either 16 or 32 bits as indicated by the operation. Addition and subtraction of two's complement integers ignore any overflow condition. This allows numbers treated as unsigned to produce the same results as if the numbers had been treated as signed.

block

The 1024 bytes of data from mass storage which are referenced by block numbers in the range {0..the number of blocks available -1}. The actual amount of data transferred and the translation from block number to device and physical record is a function of the implementation. See: "block buffer" "mass storage"

block buffer

A 1024-byte memory area where a block is made temporarily available for use. Block buffers are uniquely assigned to blocks. See: "9.7 Multiprogramming Impact"

5. DEFINITIONS OF TERMS

byte

An assembly of 8 bits. In reference to memory, it is the storage capacity for 8 bits.

character

A 7-bit number the signification of which is given by the ASCII standard. When contained in a larger field, the higher order bits are zero. See: "6. REFERENCES"

compilation

The action of converting text words from the input stream into an internal form suitable for later execution. When in the compile state, the compilation addresses of FORTH words are compiled into the dictionary for later execution by the address interpreter. Numbers are compiled to be placed on the data stack when later executed. Numbers are accepted from the input stream unsigned or negatively signed and converted using the value of BASE. See: "number" "number conversion" "interpreter, text"

defining word

A word that, when executed, creates a new dictionary entry in the compilation vocabulary. The new word name is taken from the input stream. If the input stream is exhausted before the new name is available, an error condition exists. Examples of defining words are: : CONSTANT CREATE

definition

See: "word definition"

dictionary

A structure of word definitions in computer memory which is extensible and grows toward higher memory addresses. Entries are organized in vocabularies to aid location by name. See: "search order"

display

The process of sending one or more characters to the current output device. These characters are typically displayed or printed on a terminal. The selection of the current output device is system dependent.

division, floored

Integer division in which the remainder carries the sign of the divisor or is zero, and the quotient is rounded to its arithmetic floor. Note that, except for error conditions, `n1 n2 SWAP OVER /MOD ROT * +` is identical to `n1`. See: "floor, arithmetic"

Examples:

dividend	divisor	remainder	quotient
10	7	3	1
-10	7	4	-2
10	-7	-4	-2
-10	-7	-3	1

5. DEFINITIONS OF TERMS

equivalent execution

A standard program will produce the same results, exclusive of timing dependencies, when given the same inputs on any Standard System which has sufficient resources to execute the program. Only standard source programs are transportable.

error condition

An exceptional condition which requires action by the system which may be other than the expected function. Refer to the section "10. Error Conditions".

false

A zero number represents the false state of a flag.

flag

A number that may have one of two logical states, false or true.
See: "false" "true"

floor, arithmetic

If z is any real number, then the floor of z is the greatest integer less than or equal to z .

The floor of $+.6$ is 0

The floor of $-.4$ is -1

free field format

Numbers are converted using the value of BASE and then displayed with no leading zeros. A trailing space is displayed. The number of characters displayed is the minimum number of characters, at least one, to uniquely represent the number. See: "number conversion"

glossary

A set of explanations in natural language to describe the corresponding computer execution of word definitions.

immediate word

A word which executes when encountered during compilation or interpretation. Immediate words handle special cases during compilation. See, for example, IF LITERAL ." etc.

input stream

A sequence of characters available to the system, for processing by the text interpreter. The input stream conventionally may be taken from the current input device (via the text input buffer) and mass storage (via a block buffer). BLK , >IN , TIB and #TIB specify the input stream. Words using or altering BLK , >IN , TIB and #TIB are responsible for maintaining and restoring control of the input stream.

The input stream extends from the offset value of >IN to the size of the input stream. If BLK is zero the input stream is contained within the area addressed by TIB and is #TIB bytes long. If BLK is non-zero the input stream is contained within the block buffer specified by BLK and is 1024 bytes long. See: "11.8 Input Text"

5. DEFINITIONS OF TERMS

interpreter, address

The machine code instructions, routine or other facilities that execute compiled word definitions containing compilation addresses.

interpreter, text

The word definition(s) that repeatedly accepts a word name from the input stream, locates the corresponding compilation address and starts the address interpreter to execute it. Text from the input stream interpreted as a number leaves the corresponding value on the data stack. Numbers are accepted from the input stream unsigned or negatively signed and converted using the value of BASE. See: "number" "number conversion"

layers

The grouping of word names of each Standard word set to show like characteristics. No implementation requirements are implied by this grouping.

layer, compiler

Word definitions which add new procedures to the dictionary or which aid compilation by adding compilation addresses or data structures to the dictionary.

layer, devices

Word definitions which allow access to mass storage and computer peripheral devices.

layer, interpreter

Word definitions which support vocabularies, terminal output, and the interpretation of text from the text input buffer or a mass storage device by executing the corresponding word definitions.

layer, nucleus

Word definitions generally defined in machine code that control the execution of the fundamental operations of a virtual FORTH machine. This includes the address interpreter.

load

Redirection of the text interpreter's input stream to be from mass storage. This is the general method for compilation of new definitions into the dictionary.

mass storage

Storage which might reside outside FORTH's address space. Mass storage data is made available in the form of 1024-byte blocks. A block is accessible within the FORTH address space in a block buffer. When a block has been indicated as UPDAted (modified) the block will ultimately be transferred to mass storage.

5. DEFINITIONS OF TERMS

number

When values exist within a larger field, the most-significant bits are zero. 16-bit numbers are represented in memory by addressing the first of two bytes at consecutive addresses. The byte order is unspecified by this Standard. Double numbers are represented on the stack with the most-significant 16 bits (with sign) most accessible. Double numbers are represented in memory by two consecutive 16-bit numbers. The address of the least-significant 16 bits is two greater than the address of the most-significant 16 bits. The byte order within each 16-bit field is unspecified. See: "arithmetic, two's complement" "number types" "9.8 Numbers" "11.7 Stack Parameters"

number conversion

Numbers are maintained internally in binary and represented externally by using graphic characters within the ASCII character set. Conversion between the internal and external forms is performed using the current value of BASE to determine the digits of a number. A digit has a value ranging from zero to the value of BASE-1. The digit with the value zero is represented by the ASCII character "0" (position 3/0 with the decimal equivalent of 48). This representation of digits proceeds through the ASCII character set to the character "9" corresponding to the decimal value 9. For digits with a value exceeding 9, the ASCII graphic characters beginning with the character "A" (position 4/1 with the decimal equivalent 65) corresponding to the decimal value 10 are used. This sequence then continues up to and including the digit with the decimal value 71 which is represented by the ASCII character "~" (position 7/14 with a decimal equivalent 126). A negative number may be represented by preceding the digits with a single leading minus sign, the character "-".

number types

All number types consist of some number of bits. These bits are either arbitrary or are weighted.

Signed and unsigned numbers use weighted bits. Weighted bits within a number have a value of a power of two beginning with the rightmost (least-significant) bit having the value of two to the zero power. This weighting continues to the leftmost bit increasing the power by one for each bit. For an unsigned number this weighting pattern includes the leftmost bit; thus, for an unsigned 16-bit number the weight of the leftmost bit is 32,768. For a signed number this weighting pattern includes the leftmost bit but the weight of the leftmost bit is negated; thus, for a signed 16-bit number the weight of the leftmost bit is -32,768. This weighting pattern for signed numbers is called two's complement notation.

Unspecified weighted numbers are either unsigned numbers or signed numbers; program context determines whether the number is signed or unsigned. See: "11.7 Stack Parameters"

5. DEFINITIONS OF TERMS

pictured numeric output

The use of numeric output definitions which convert numerical values into text strings. These definitions are used in a sequence which resembles a symbolic 'picture' of the desired text format. Conversion proceeds from least-significant digit to most-significant digit, and converted characters are stored from higher memory addresses to lower.

program

A complete specification of execution to achieve a specific function (application task) expressed in FORTH source code form.

receive

The process of obtaining one character from the current input device. The selection of the current input device is system dependent.

recursion

The process of self-reference, either directly or indirectly.

return

The means of indicating the end of text by striking a key on an input device. The key used is system dependent. This key is typically called "RETURN", "CARRIAGE RETURN", or "ENTER".

screen

Textual data arranged for editing. By convention, a screen consists of 16 lines (numbered 8 through 15) of 64 characters each. Screens usually contain program source text, but may be used to view mass storage data. The first byte of a screen occupies the first byte of a mass storage block, which is the beginning point for text interpretation during a load.

search order

A specification of the order in which selected vocabularies in the dictionary are searched. Execution of a vocabulary makes it the first vocabulary in the search order. The dictionary is searched whenever a word is to be located by its name. This order applies to all dictionary searches unless otherwise noted. The search order begins with the last vocabulary executed and ends with FORTH, unless altered in a system dependent manner.

source definition

Text consisting of word names suitable for compilation or execution by the text interpreter. Such text is usually arranged in screens and maintained on a mass storage device.

stack, data

A last in, first out list consisting of 16-bit binary values. This stack is primarily used to hold intermediate values during execution of word definitions. Stack values may represent numbers, characters, addresses, boolean values, etc.

When the name 'stack' is used alone, it implies the data stack.

5. DEFINITIONS OF TERMS

stack, return

A last in, first out list which contains the addresses of word definitions whose execution has not been completed by the address interpreter. As a word definition passes control to another definition, the return point is placed on the return stack.

The return stack may cautiously be used for other values.

string, counted

A sequence of consecutive 8-bit bytes located in memory by their low memory address. The byte at this address contains a count {0..255} of the number of bytes following which are part of the string. The count does not include the count byte itself. Counted strings usually contain ASCII characters.

string, text

A sequence of consecutive 8-bit bytes located in memory by their low memory address and length in bytes. Strings usually, but not exclusively, contain ASCII characters. When the term 'string' is used alone or in conjunction with other words it refers to text strings.

structure, control

A group of FORTH words which when executed alter the execution sequence. The group starts and terminates with compiler words. Examples of control structures: DO ... LOOP DO ... +LOOP BEGIN ... WHILE ... REPEAT BEGIN ... UNTIL IF ... THEN IF ... ELSE ... THEN See: "9.9 Control Structures"

transportability

This term indicates that equivalent execution results when a program is executed on other than the system on which it was created.
See: "equivalent execution"

true

A non-zero value represents the true state of a flag. Any non-zero value will be accepted by a standard word as 'true'; all standard words return a 16-bit value with all bits set to one when returning a 'true' flag.

user area

An area in memory which contains the storage for user variables.

variable, user

A variable whose data storage area is usually located in the user area. Some system variables are maintained in the user area so that the words may be re-entrant to different users.

vocabulary

An ordered list of word definitions. Vocabularies are an advantage in separating different word definitions that may have the same name. More than one definition with the same name can exist in one vocabulary. The latter is called a redefinition. The most recently created redefinition will be found when the vocabulary is searched.

5. DEFINITIONS OF TERMS

vocabulary, compilation

The vocabulary into which new word definitions are appended.

word

A sequence of characters terminated by one blank or the end of the input stream. Leading blanks are ignored. Words are usually obtained via the input stream.

word definition

A named FORTH execution procedure compiled into the dictionary. Its execution may be defined in terms of machine code, as a sequence of compilation addresses, or other compiled words.

word name

The name of a word definition. Word names are limited to 31 characters and may not contain an ASCII space. If two definitions have different word names in the same vocabulary they must be uniquely findable when this vocabulary is searched. See: "vocabulary" "9.5.3 EXPECT"

word set

A named group of FORTH word definitions in the Standard.

word set, assembler extension

Additional words which facilitate programming in the native machine language of the computer which are by nature system dependent.

word set, double number extension

Additional words which facilitate manipulation of 32-bit numbers.

word set, required

The minimum words needed to compile and execute Standard Programs.

word set, system extension

Additional words which facilitate the access to internal system characteristics.

word, standard

A named FORTH procedure definition, in the Required word set or any extension word sets, formally reviewed and accepted by the Standards Team.

6. REFERENCES

6. REFERENCES

The following document is considered to be a portion of this Standard:

American National Standard Code for Information Interchange, X3.4-1977
(ASCII), American National Standards Institute, 1430 Broadway, New York,
NY 10018, USA.

The following documents are noted as pertinent to the FORTH-83 Standard, but
are not part of this Standard.

FORTH-77, FORTH Users Group, FST-780314

FORTH-78, FORTH International Standards Team

FORTH-79, FORTH Standards Team

FORTH-83 STANDARD, Appendices, FORTH Standards Team

Webster's Collegiate Dictionary shall be used to resolve conflicts in
spelling and English word usage.

7. REQUIREMENTS**7. REQUIREMENTS****7.1 Documentation Requirements**

7.1.1 Each Standard System shall be accompanied by a statement of:

- 1. System dictionary space used in bytes;**
- 2. Application dictionary space available in bytes;**
- 3. Data stack space in bytes;**
- 4. Return stack space in bytes;**
- 5. Mass storage block ranges used by the system;**
- 6. Mass storage block ranges available to applications;**
- 7. Operator's terminal facilities available;**
- 8. System action taken upon each of the general or specified error conditions as identified in this Standard.**

7.1.2 Each Standard Program shall be accompanied by a statement of the minimum requirements for:

- 1. Dictionary space in bytes;**
- 2. Data stack space in bytes;**
- 3. Return stack space in bytes;**
- 4. Mass storage block ranges;**
- 5. Operator's terminal facilities.**

7. REQUIREMENTS**7.2 Testing Requirements**

The following host computer configuration is specified as a minimum environment for testing against this Standard. Applications may require different capacities.

1. 2000 bytes of memory for application dictionary;
2. Data stack of 64 bytes;
3. Return stack of 48 bytes;
4. Mass storage capacity of 32 blocks, numbered 0 through 31;
5. One ASCII input/output device acting as an operator's terminal.

8. COMPLIANCE AND LABELING

8. COMPLIANCE AND LABELING

The FORTH Standards Team hereby specifies the requirements for labeling of systems and applications so that the conditions for program portability may be established.

A Standard System may use the specified labeling if it complies with the terms of this Standard and meets the particular Word Set definitions.

A Standard Program (application) may use the specified labeling if it utilizes the specified Standard System according to this Standard and executes equivalently on any such system.

In a system or application, a standard word may not be redefined to perform a different function within the vocabulary FORTH.

FORTH Standard

A system may be labeled:

FORTH-83 Standard

if it includes all of the Required Word Set in either source or object form and complies with the text of this Standard. After executing "FORTH-83" the dictionary must contain all of the Required Word Set in the vocabulary FORTH, as specified in this Standard.

Standard Sub-set

A system may be labeled:

FORTH-83 Standard Sub-set

if it includes a portion of the Required Word Set and complies with the remaining text of this Standard. However, no Required Word may be present with a non-standard definition.

8. COMPLIANCE AND LABELING**Standard with Extensions**

A system may be labeled:

FORTH-83 Standard with <name> Standard Extension(s)

if it comprises a FORTH-83 Standard System and one or more Standard Extension Word Set(s). For example, a designation would be in the form:

FORTH-83 Standard with Double-Number Standard Extension

Standard Program

A FORTH source program which executes equivalently on any Standard System may be labeled:

FORTH-83 Standard Program

See: "equivalent execution" "7. REQUIREMENTS"

Standard Program with Environmental Dependencies

A program which is standard in all ways except for specific environmentally dependent words may be labeled:

FORTH-83 Standard Program with Environmental Dependencies

if the following additional requirements are met:

- 1) Environmental dependencies (including hardware dependencies) shall be factored into an isolated set of application word definitions.
- 2) Each environmentally dependent word definition must be fully documented, including all dependencies in a manner at least as detailed as the standard words.

9. USAGE

9. USAGE

9.1 Words Names and Word Definitions

A Standard Program may reference only the definitions of the Required Word Set and Standard Extensions and definitions which are subsequently defined in terms of these words. Furthermore, a Standard Program must use the standard words as required by any conventions of this Standard. Equivalent execution must result from Standard Programs.

The implementation of a Standard System may use words and techniques outside the scope of the Standard, provided that no program running on that system is required to use words outside the Standard for normal operation.

If a Standard System or Standard Program redefines Standard definitions within the FORTH vocabulary, these definitions must comply with the Standard.

9.2 Addressable Memory

The FORTH system may share the dictionary space with the user's application. The native addressing protocol of the host computer is beyond the scope of this Standard.

Therefore, in a Standard Program, the user may only operate on data which was stored by the application. No exceptions!

A Standard Program may address:

1. parameter fields of words created with CREATE , VARIABLE , and user defined words which execute CREATE ;
2. dictionary space ALLOTTed;
3. data in a valid mass storage block buffer,
See: "9.7 Multiprogramming Impact";
4. data area of user variables;
5. text input buffer and PAD up to the amount specified as the minimum for each area.

9. USAGE

A Standard Program may NOT address:

1. directly into the data or return stacks;
2. into a definition's name field, link field, or code field;
3. into a definition's parameter field if not stored by the application.

9.3 Return Stack

A Standard Program may cautiously use the return stack with the following restrictions:

The return stack may not be accessed inside a do-loop for values placed on the return stack before the loop was entered. Further, neither I nor J may be used to obtain the index of a loop if values are placed and remain on the return stack within the loop. When the do-loop is executed all values placed on the return stack within that loop must be removed before LOOP, +LOOP, or LEAVE is executed. Similarly, all values placed on the return stack within a colon definition must be removed before the colon definition is terminated at ; or before EXIT is executed.

9.4 Compilation

The system uses the return stack and the dictionary in a system dependent manner during the compilation of colon definitions. Some words use the data stack in a system dependent manner during compilation.

See: "sys (11.7)"

9.5 Terminal Input and Output

9.5.1 KEY

A Standard System must receive all valid ASCII characters. Each KEY receives one ASCII character, with more-significant bits environmentally dependent and might not be zero. KEY must receive as many bits as are obtainable. A Standard Program without environmental dependencies may only use the least-significant 7-bit ASCII character received by KEY. For example:

KEY 127 AND

9.5.2 EXPECT

Control characters may be processed to allow system dependent editing of the characters prior to receipt. Therefore, a Standard Program may not anticipate that control characters can be received.

9. USAGE

9.5.3 EMIT

Because of the potential non-transportable action by terminal devices of control characters, the use of ASCII control characters is an environmental dependency. Each EMIT deals with only one ASCII character. The ASCII character occupies the least-significant 7 bits; the more-significant bits may be environmentally dependent. Using the more-significant bits when other than zero is an environmentally dependent usage. EMIT must display as many bits as can be sent.

9.5.4 TYPE

Because of the potential non-transportable action by terminal devices of control characters, the use of ASCII control characters is an environmental dependency.

9.6 Transporting Programs Between Standard Systems

Further usage requirements are expected to be added for transporting programs between Standard Systems.

9.7 Multiprogramming Impact

In a multiprogrammed system, Device Layer words and those words which implicitly reference the Device Layer words may relinquish control of the processor to other tasks. Although there is insufficient experience to specify a standard for multiprogramming, historical usage dictates that a programmer be aware of the potential impact with regard to resources shared between tasks. The only shared resources specified within the Standard are block buffers. Therefore the address of a block buffer returned by BLOCK or BUFFER becomes invalid during and after the execution of any word marked by the attribute M in the glossary or any words executing them. A block buffer is valid only if its address is valid. See: "11.4 Attributes"

9.8 Numbers

Interpreted or compiled numbers are in the range {-32,768..65,535}.
See: "number conversion"

9.9 Control Structures

Control structures are compiled inside colon definitions. Control structures can be nested but cannot overlap. For additional limitations see DO .

10. ERROR CONDITIONS**10. ERROR CONDITIONS****10.1 Possible Actions on an Error**

When an error condition occurs, a Standard System may take one or more of the following actions:

1. ignore and continue;
2. display a message;
3. execute a particular word;
4. set interpret state and interpret a block;
5. set interpret state and begin interpretation;
6. other system dependent actions.

See: "7.1 Documentation Requirements"

10. ERROR CONDITIONS

10.2 General Error Conditions

The following error conditions apply in many situations. These error conditions are listed below, but may occur at various times and with various words.

1. input stream exhausted before encountering a required `<name>` or delimiting character;
2. insufficient stack space or insufficient number of stack entries during text interpretation or compilation;
3. a word not found and not a valid number, during text interpretation or compilation;
4. compilation of incorrectly nested control structures;
5. execution of words restricted to compilation only, when not in the compile state and while not compiling a colon definition;
6. FORGETting within the system to a point that removes a word required for correct execution;
7. insufficient space remaining in the dictionary;
8. a stack parameter out of range, e.g., a negative number when a +n was specified in the glossary;
9. correct mass storage read or write was not possible.

11. GLOSSARY NOTATION

11. GLOSSARY NOTATION

11.1 Order

The glossary definitions are listed in ASCII alphabetical order.

11.2 Capitalization

Word names are capitalized throughout this Standard.

11.3 Stack Notation

The stack parameters input to and output from a definition are described using the notation:

before -- after

before stack parameters before execution
after stack parameters after execution

In this notation, the top of the stack is to the right. Words may also be shown in context when appropriate.

Unless otherwise noted, all stack notation describes execution time. If it applies at compile time, the line is followed by: (compiling) .

11.4 Attributes

Capitalized symbols indicate attributes of the defined words:

C The word may only be used during compilation of a colon definition.

I Indicates that the word is IMMEDIATE and will execute during compilation, unless special action is taken.

M This word has a potential multiprogramming impact.
See: "9.7 Multiprogramming Impact"

U A user variable.

11. GLOSSARY NOTATION

11.5 Serial Numbers

When a substantive alteration to a word's definition is made or when a new word is added, the serial number will be the last two digits of the year of the Standard in which such change was made (i.e., "83"). When such change is made within a Working Draft, the number will be suffixed with the character identifying the draft (i.e., "83A").

11.6 Pronunciation

The natural language pronunciation of word names is given in double quotes (") where it differs from English pronunciation.

11.7 Stack Parameters

Unless otherwise stated, all references to numbers apply to 16-bit signed integers. The implied range of values is shown as {from..to}. The content of an address is shown by double braces, particularly for the contents of variables, i.e., BASE {{2..72}}.

The following are the stack parameter abbreviations and types of numbers used throughout the glossary. These abbreviations may be suffixed with a digit to differentiate multiple parameters of the same type.

Stack Abbrev.	Number Type	Range in Decimal	Minimum Field
flag	boolean	0=false, else=true	16
true	boolean	-1 (as a result)	16
false	boolean	0	16
b	bit	{0..1}	1
char	character	{0..127}	7
8b	8 arbitrary bits (byte)	not applicable	8
16b	16 arbitrary bits	not applicable	16
n	number (weighted bits)	{-32,768..32,767}	16
+n	positive number	{0..32,767}	16
u	unsigned number	{0..65,535}	16
w	unspecified weighted number (n or u)	{-32,768..65,535}	16
addr	address (same as u)	{0..65,535}	16
32b	32 arbitrary bits	not applicable	32
d	double number	{-2,147,483,648.. 2,147,483,647}	32
+d	positive double number	{0..2,147,483,647}	32
ud	unsigned double number	{0..4,294,967,295}	32
wd	unspecified weighted double number (d or ud)	{-2,147,483,648.. 4,294,967,295}	32
sys	0, 1, or more system dependent stack entries	not applicable	na

11. GLOSSARY NOTATION

Any other symbol refers to an arbitrary signed 16-bit integer in the range $\{-32,768..32,767\}$, unless otherwise noted.

Because of the use of two's complement arithmetic, the signed 16-bit number $(n) - 1$ has the same bit representation as the unsigned number $(u) 65,535$. Both of these numbers are within the set of unspecified weighted numbers (w) . See: "arithmetic, two's complement" "number" "number types" "stack, data"

11.8 Input Text

<name>

An arbitrary FORTH word accepted from the input stream. This notation refers to text from the input stream, not to values on the data stack. See: "10.2 General Error Conditions"

qqq

A sequence of arbitrary characters accepted from the input stream until the first occurrence of the specified delimiting character. The delimiter is accepted from the input stream, but is not one of the characters qqq and is therefore not otherwise processed. This notation refers to text from the input stream, not to values on the data stack. Unless noted otherwise, the number of characters accepted may be from 0 to 255. See: "10.2 General Error Conditions"

11.9 References to other words and definitions

Glossary definitions may refer to other glossary definitions or to definitions of terms. Such references are made using the expression "See:". These references provide additional information which apply as if the information is a portion of the glossary entry using "See:".

12. REQUIRED WORD SET

12. REQUIRED WORD SET

12.1 The Required Word Set Layers

The words of the Required Word Set are grouped to show like characteristics. No implementation requirements should be inferred from this grouping.

Nucleus layer

```
! * */ */MOD + +! - / /MOD Ø< Ø= Ø> 1+ 1-
2+ 2- 2/ < = > >R ?DUP @ ABS AND C! C@ CMOVE
CMOVE> COUNT D+ D< DEPTH DNEGATE DROP DUP EXECUTE EXIT
FILL I J MAX MIN MOD NEGATE NOT OR OVER PICK R>
R@ ROLL ROT SWAP U< UM* UM/MOD XOR
```

Device layer

```
BLOCK BUFFER CR EMIT EXPECT FLUSH KEY SAVE-BUFFERS
SPACE SPACES TYPE UPDATE
```

Interpreter layer

```
# #> #S #TIB ' ( -TRAILING . .( <# >BODY >IN
ABORT BASE BLK CONVERT DECIMAL DEFINITIONS FIND FORGET
FORTH FORTH-83 HERE HOLD LOAD PAD QUIT SIGN SPAN TIB
U. WORD
```

Compiler layer

```
+LOOP , ." : ; ABORT" ALLOT BEGIN COMPIL CONSTANT
CREATE DO DOES> ELSE IF IMMEDIATE LEAVE LITERAL LOOP
REPEAT STATE THEN UNTIL VARIABLE VOCABULARY WHILE [ [']
[COMPILE] ]
```

12. REQUIRED WORD SET

12.2 The Required Word Set Glossary

!	16b addr -- 16b is stored at addr.	79	"store"
#	+d1 -- +d2 The remainder of +d1 divided by the value of BASE is converted to an ASCII character and appended to the output string toward lower memory addresses. +d2 is the quotient and is maintained for further processing. Typically used between <# and #> .	79	"sharp"
#>	32b -- addr +n Pictured numeric output conversion is ended dropping 32b. addr is the address of the resulting output string. +n is the number of characters in the output string. addr and +n together are suitable for TYPE .	79	"sharp-greater"
#S	+d -- 0 0 +d is converted appending each resultant character into the pictured numeric output string until the quotient (see: #) is zero. A single zero is added to the output string if the number was initially zero. Typically used between <# and #> .	79	"sharp-s"
#TIB	-- addr The address of a variable containing the number of bytes in the text input buffer. #TIB is accessed by WORD when BLK is zero. {{0..capacity of TIB}} See: "input stream"	U,83	"number-t-i-b"
'	-- addr Used in the form: ' <name> addr is the compilation address of <name>. An error condition exists if <name> is not found in the currently active search order.	M,83	"tick"
(-- -- (compiling) Used in the form: (<u>qqq</u>) The characters <u>qqq</u> , delimited by) (closing parenthesis), are considered comments. Comments are not otherwise processed. The blank following (is not part of <u>qqq</u> . (may be freely used while interpreting or compiling. The number of characters in <u>qqq</u> may be from zero to the number of characters remaining in the input stream up to the closing parenthesis.	I,M,83	"paren"
*	w1 w2 -- w3 w3 is the least-significant 16 bits of the arithmetic product of w1 times w2.	79	"times"

12. REQUIRED WORD SET

."	-- -- (compiling)	C,I,83	"dot-quote"
Used in the form: ." <u>ccc</u> " Later execution will display the characters <u>ccc</u> up to but not including the delimiting " (close-quote). The blank following ." is not part of <u>ccc</u> .			
.(-- -- (compiling)	I,M,83	"dot-paren"
Used in the form: .(<u>ccc</u>) The characters <u>ccc</u> up to but not including the delimiting) (closing parenthesis) are displayed. The blank following .(is not part of <u>ccc</u> .			
/	n1 n2 -- n3	83	"divide"
n3 is the floor of the quotient of n1 divided by the divisor n2. An error condition results if the divisor is zero or if the quotient falls outside of the range {-32,768..32,767}. See: "division, floored"			
/MOD	n1 n2 -- n3 n4	83	"divide-mod"
n3 is the remainder and n4 the floor of the quotient of n1 divided by the divisor n2. n3 has the same sign as n2 or is zero. An error condition results if the divisor is zero or if the quotient falls outside of the range {-32,768..32,767}. See: "division, floored"			
0<	n -- flag	83	"zero-less"
flag is true if n is less than zero (negative).			
0=	w -- flag	83	"zero-equals"
flag is true if w is zero.			
0>	n -- flag	83	"zero-greater"
flag is true if n is greater than zero.			
1+	w1 -- w2	79	"one-plus"
w2 is the result of adding one to w1 according to the operation of + .			
1-	w1 -- w2	79	"one-minus"
w2 is the result of subtracting one from w1 according to the operation of - .			
2+	w1 -- w2	79	"two-plus"
w2 is the result of adding two to w1 according to the operation of + .			
2-	w1 -- w2	79	"two-minus"
w2 is the result of subtracting two from w1 according to the operation of - .			
2/	n1 -- n2	83	"two-divide"
n2 is the result of arithmetically shifting n1 right one bit. The sign is included in the shift and remains unchanged.			

12. REQUIRED WORD SET

:	-- sys	M,79	"colon"
A defining word executed in the form: : <name> ... ; Create a word definition for <name> in the compilation vocabulary and set compilation state. The search order is changed so that the first vocabulary in the search order is replaced by the compilation vocabulary. The compilation vocabulary is unchanged. The text from the input stream is subsequently compiled. <name> is called a "colon definition". The newly created word definition for <name> cannot be found in the dictionary until the corresponding ; or ;CODE is successfully processed. An error condition exists if a word is not found and cannot be converted to a number or if, during compilation from mass storage, the input stream is exhausted before encountering ; or ;CODE . sys is balanced with its corresponding ; . See: "compilation" "9.4 Compilation"			
;	-- sys -- (compiling)	C,I,79	"semi-colon"
Stops compilation of a colon definition, allows the <name> of this colon definition to be found in the dictionary, sets interpret state and compiles EXIT (or a system dependent word which performs an equivalent function). sys is balanced with its corresponding : . See: EXIT : "stack, return" "9.4 Compilation"			
<	n1 n2 -- flag	83	"less-than"
flag is true if n1 is less than n2. -32768 32767 < must return true. -32768 0 < must return true.			
<#	--	79	"less-sharp"
Initialize pictured numeric output conversion. The words: # #> #S <# HOLD SIGN can be used to specify the conversion of a double number into an ASCII text string stored in right-to-left order.			
=	w1 w2 -- flag	83	"equals"
flag is true if w1 is equal to w2.			
>	n1 n2 -- flag	83	"greater-than"
flag is true if n1 is greater than n2. -32768 32767 > must return false -32768 0 > must return false			
>BODY	addr1 -- addr2	83	"to-body"
addr2 is the parameter field address corresponding to the compilation address addr1. See: "9.2 Addressable Memory"			
>IN	-- addr	U,79	"to-in"
The address of a variable which contains the present character offset within the input stream [{0..the number of characters in the input stream}]. See: WORD			

12. REQUIRED WORD SET

>R 16b -- C,79 "to-r"
Transfers 16b to the return stack. See: "9.3 Return Stack"

?DUP 16b -- 16b 16b 79 "question-dupe"
 or 0 -- 0
Duplicate 16b if it is non-zero.

@ addr -- 16b 79 "fetch"
16b is the value at addr.

ABORT 79
Clears the data stack and performs the function of QUIT . No message is displayed.

ABORT" flag -- C,I,83 "abort-quote"
 -- (compiling)
Used in the form:
 flag ABORT" qqq"
When later executed, if flag is true the characters qqq, delimited by " (close-quote), are displayed and then a system dependent error abort sequence, including the function of ABORT , is performed. If flag is false, the flag is dropped and execution continues. The blank following ABORT" is not part of qqq.

ABS n -- u 79 "absolute"
u is the absolute value of n. If n is -32,768 then u is the same value.
See: "arithmetic, two's complement"

ALLOT w -- 79
Allocates w bytes in the dictionary. The address of the next available dictionary location is updated accordingly.

AND 16b1 16b2 -- 16b3 79
16b3 is the bit-by-bit logical 'and' of 16b1 with 16b2.

BASE -- addr U,83
The address of a variable containing the current numeric conversion radix. {{2..72}}

BEGIN -- C,I,79
 -- sys (compiling)
Used in the form:
 BEGIN ... flag UNTIL
 or
 BEGIN ... flag WHILE ... REPEAT
BEGIN marks the start of a word sequence for repetitive execution. A BEGIN-UNTIL loop will be repeated until flag is true. A BEGIN-WHILE-REPEAT loop will be repeated until flag is false. The words after UNTIL or REPEAT will be executed when either loop is finished. sys is balanced with its corresponding UNTIL or WHILE . See: "9.9 Control Structures"

12. REQUIRED WORD SET

BLK	-- addr	U,79	"b-l-k"
<p>The address of a variable containing the number of the mass storage block being interpreted as the input stream. If the value of BLK is zero the input stream is taken from the text input buffer. {{0..the number of blocks available -1}} See: TIB "input stream".</p>			

BLOCK u -- addr M,83

addr is the address of the assigned buffer of the first byte of block u. If the block occupying that buffer is not block u and has been UPDATeEd it is transferred to mass storage before assigning the buffer. If block u is not already in memory, it is transferred from mass storage into an assigned block buffer. A block may not be assigned to more than one buffer. If u is not an available block number, an error condition exists. Only data within the last buffer referenced by BLOCK or BUFFER is valid. The contents of a block buffer must not be changed unless the change may be transferred to mass storage.

BUFFER u -- addr M,83
Assigns a block buffer to block u. addr is the address of the first byte of the block within its buffer. This function is fully specified by the definition for BLOCK except that if the block is not already in memory it might not be transferred from mass storage. The contents of the block buffer assigned to block u by BUFFER are unspecified.

C1	16b addr --	79	"c-store"
	The least-significant 8 bits of 16b are stored into the byte at addr.		

Ce	addr -- 8b	79	"c-fetch"
	8b is the contents of the byte at addr.		

```
CMOVE      addr1 addr2 u --                        83      "c-move"
Move u bytes beginning at address addr1 to addr2.  The byte at addr1 is
moved first, proceeding toward high memory.  If u is zero nothing is
moved.
```

```
CMOVE>      addr1 addr2 u --      83      "c-move-up"
Move the u bytes at address addr1 to addr2.  The move begins by moving
the byte at (addr1 plus u minus 1) to (addr2 plus u minus 1) and proceeds
to successively lower addresses for u bytes.  If u is zero nothing is
moved.  (Useful for sliding a string towards higher addresses).
```

COMPILE --- C,83

Typically used in the form:

: <name> ... COMPILE <namex> ... ;

When <name> is executed, the compilation address compiled for <namex> is
compiled and not executed. <name> is typically immediate and <namex> is
typically not immediate. See: "compilation"

CONSTANT 16b — M,83

A defining word executed in the form:

16b CONSTANT <name>

Creates a dictionary entry for <name> so that when <name> is later executed, 16b will be left on the stack.

12. REQUIRED WORD SET

CONVERT +d1 addr1 -- +d2 addr2 79
 +d2 is the result of converting the characters within the text beginning at addr1+1 into digits, using the value of BASE, and accumulating each into +d1 after multiplying +d1 by the value of BASE. Conversion continues until an unconvertible character is encountered. addr2 is the location of the first unconvertible character.

COUNT addr1 -- addr2 +n 79
 addr2 is addr1+1 and +n is the length of the counted string at addr1. The byte at addr1 contains the byte count +n. Range of +n is {0..255}. See: "string, counted"

CR -- M,79 "c-r"
 Displays a carriage-return and line-feed or equivalent operation.

CREATE -- M,79
 A defining word executed in the form:
 CREATE <name>
 Creates a dictionary entry for <name>. After <name> is created, the next available dictionary location is the first byte of <name>'s parameter field. When <name> is subsequently executed, the address of the first byte of <name>'s parameter field is left on the stack. CREATE does not allocate space in <name>'s parameter field.

D+ wd1 wd2 -- wd3 79 "d-plus"
 wd3 is the arithmetic sum of wd1 plus wd2.

D< d1 d2 -- flag 83 "d-less-than"
 flag is true if d1 is less than d2 according to the operation of < except extended to 32 bits.

DECIMAL -- 79
 Set the input-output numeric conversion base to ten.

DEFINITIONS -- 79
 The compilation vocabulary is changed to be the same as the first vocabulary in the search order. See: "vocabulary, compilation"

DEPTH -- +n 79
 +n is the number of 16-bit values contained in the data stack before +n was placed on the stack.

DNEGATE d1 -- d2 79 "d-negate"
 d2 is the two's complement of d1.

12. REQUIRED WORD SET

DO w1 w2 -- C,I,83
 -- sys (compiling)

Used in the form:

DO ... LOOP

or

DO ... +LOOP

Begins a loop which terminates based on control parameters. The loop index begins at w2, and terminates based on the limit w1. See LOOP and +LOOP for details on how the loop is terminated. The loop is always executed at least once. For example: w DUP DO ... LOOP executes 65,536 times. sys is balanced with its corresponding LOOP or +LOOP .
 See: "9.9 Control Structures"

An error condition exists if insufficient space is available for at least three nesting levels.

DOES> -- addr C,I,83 "does"
 -- (compiling)

Defines the execution-time action of a word created by a high-level defining word. Used in the form:

: <namex> ... <create> ... DOES> ... ;

and then

<namex> <name>

where <create> is CREATE or any user defined word which executes CREATE .

Marks the termination of the defining part of the defining word <namex> and then begins the definition of the execution-time action for words that will later be defined by <namex>. When <name> is later executed, the address of <name>'s parameter field is placed on the stack and then the sequence of words between DOES> and ; are executed.

DROP 16b -- 79
 16b is removed from the stack.

DUP 16b -- 16b 16b 79 "dupe"
 Duplicate 16b.

ELSE -- C,I,79
 sys1 -- sys2 (compiling)

Used in the form:

flag IF ... ELSE ... THEN

ELSE executes after the true part following IF . ELSE forces execution to continue at just after THEN . sys1 is balanced with its corresponding IF . sys2 is balanced with its corresponding THEN . See: IF THEN

EMIT 16b -- M,83
 The least-significant 7-bit ASCII character is displayed. See: "9.5.3
 EMIT"

EXECUTE addr -- 79
 The word definition indicated by addr is executed. An error condition exists if addr is not a compilation address.

12. REQUIRED WORD SET

- EXIT** — C,79
Compiled within a colon definition such that when executed, that colon definition returns control to the definition that passed control to it by returning control to the return point on the top of the return stack. An error condition exists if the top of the return stack does not contain a valid return point. May not be used within a do-loop.
See: ; "stack, return" "9.3 Return Stack"
- EXPECT** addr +n — M,83
Receive characters and store each into memory. The transfer begins at addr proceeding towards higher addresses one byte per character until either a "return" is received or until +n characters have been transferred. No more than +n characters will be stored. The "return" is not stored into memory. No characters are received or transferred if +n is zero. All characters actually received and stored into memory will be displayed, with the "return" displaying as a space. See: SPAN "9.5.2 EXPECT"
- FILL** addr u 8b — 83
u bytes of memory beginning at addr are set to 8b. No action is taken if u is zero.
- FIND** addr1 — addr2 n 83
addr1 is the address of a counted string. The string contains a word name to be located in the currently active search order. If the word is not found, addr2 is the string address addr1, and n is zero. If the word is found, addr2 is the compilation address and n is set to one of two non-zero values. If the word found has the immediate attribute, n is set to one. If the word is non-immediate, n is set to minus one (true).
- FLUSH** — M,83
Performs the function of SAVE-BUFFERS then unassigns all block buffers. (This may be useful for mounting or changing mass storage media).
- FORGET** — M,83
Used in the form:
FORGET <name>
If <name> is found in the compilation vocabulary, delete <name> from the dictionary and all words added to the dictionary after <name> regardless of their vocabulary. Failure to find <name> is an error condition. An error condition also exists if the compilation vocabulary is deleted. See: "10.2 General Error Conditions"
- FORTH** — 83
The name of the primary vocabulary. Execution replaces the first vocabulary in the search order with FORTH. FORTH is initially the compilation vocabulary and the first vocabulary in the search order. New definitions become part of the FORTH vocabulary until a different compilation vocabulary is established. See: VOCABULARY
- FORTH-83** — 83
Assures that a FORTH-83 Standard System is available, otherwise an error condition exists.

HERE -- addr 79
The address of the next available dictionary location.

HOLD char -- 79
char is inserted into a pictured numeric output string. Typically used between <# and #> .

I -- w C,79
w is a copy of the loop index. May only be used in the form:
DO ... I ... LOOP
or
DO ... I ... +LOOP

IF flag -- C,I,79
 -- sys (compiling)
Used in the form:
flag IF ... ELSE ... THEN
or
flag IF ... THEN
If flag is true, the words following IF are executed and the words following ELSE until just after THEN are skipped. The ELSE part is optional.

If flag is false, words from IF through ELSE , or from IF through THEN (when no ELSE is used), are skipped. sys is balanced with its corresponding ELSE or THEN . See: "9.9 Control Structures"

IMMEDIATE -- 79
Marks the most recently created dictionary entry as a word which will be executed when encountered during compilation rather than compiled.

J -- w C,79
w is a copy of the index of the next outer loop. May only be used within a nested DO-LOOP or DO+LOOP in the form, for example:
DO ... DO ... J ... LOOP ... +LOOP

KEY -- 16b M,83
The least-significant 7 bits of 16b is the next ASCII character received. All valid ASCII characters can be received. Control characters are not processed by the system for any editing purpose. Characters received by KEY will not be displayed. See: "9.5.1 KEY"

LEAVE -- C,I,83
 -- (compiling)
Transfers execution to just beyond the next LOOP or +LOOP . The loop is terminated and loop control parameters are discarded. May only be used in the form:
DO ... LEAVE ... LOOP
or
DO ... LEAVE ... +LOOP
LEAVE may appear within other control structures which are nested within the do-loop structure. More than one LEAVE may appear within a do-loop. See: "9.3 Return Stack"

```

LITERAL          -- 16b          C,I,79
                  16b -- (compiling)
Typically used in the form:
[ 16b ] LITERAL
Compiles a system dependent operation so that when later executed, 16b
will be left on the stack.

LOAD              u --          M,79
The contents of >IN and BLK , which locate the current input stream, are
saved. The input stream is then redirected to the beginning of screen u
by setting >IN to zero and BLK to u. The screen is then interpreted. If
interpretation from screen u is not terminated explicitly it will be
terminated when the input stream is exhausted and then the contents of
>IN and BLK will be restored. An error condition exists if u is zero.
See: >IN BLK BLOCK

LOOP              --          C,I,83
                  sys -- (compiling)
Increments the DO-LOOP index by one. If the new index was incremented
across the boundary between limit-1 and limit the loop is terminated and
loop control parameters are discarded. When the loop is not terminated,
execution continues to just after the corresponding DO . sys is balanced
with its corresponding DO . See: DO

MAX               n1 n2 -- n3          79          "max"
n3 is the greater of n1 and n2 according to the operation of > .

MIN               n1 n2 -- n3          79          "min"
n3 is the lesser of n1 and n2 according to the operation of < .

MOD               n1 n2 -- n3          83
n3 is the remainder after dividing n1 by the divisor n2. n3 has the same
sign as n2 or is zero. An error condition results if the divisor is zero
or if the quotient falls outside of the range {-32,768..32,767}.
See: "division, floored"

NEGATE            n1 -- n2          79
n2 is the two's complement of n1, i.e., the difference of zero less n1.

NOT               16b1 -- 16b2        83
16b2 is the one's complement of 16b1.

OR                16b1 16b2 -- 16b3   79
16b3 is the bit-by-bit inclusive-or of 16b1 with 16b2.

OVER              16b1 16b2 -- 16b1 16b2 16b3 79
16b3 is a copy of 16b1.

PAD               -- addr          83
The lower address of a scratch area used to hold data for intermediate
processing. The address or contents of PAD may change and the data lost
if the address of the next available dictionary location is changed. The
minimum capacity of PAD is 84 characters.

```

12. REQUIRED WORD SET

PICK	+n -- 16b	83	
16b is a copy of the +nth stack value, not counting +n itself. {0..the number of elements on stack-1}			
0 PICK is equivalent to DUP			
1 PICK is equivalent to OVER			
QUIT	--	79	
Clears the return stack, sets interpret state, accepts new input from the current input device, and begins text interpretation. No message is displayed.			
R>	-- 16b	C,79	"r-from"
16b is removed from the return stack and transferred to the data stack.			
See: "9.3 Return Stack"			
R@	-- 16b	C,79	"r-fetch"
16b is a copy of the top of the return stack.			
REPEAT	--	C,I,79	
sys -- (compiling)			
Used in the form:			
BEGIN ... flag WHILE ... REPEAT			
At execution time, REPEAT continues execution to just after the corresponding BEGIN . sys is balanced with its corresponding WHILE .			
See: BEGIN			
ROLL	+n --	83	
The +nth stack value, not counting +n itself is first removed and then transferred to the top of the stack, moving the remaining values into the vacated position. {0..the number of elements on the stack-1}			
2 ROLL is equivalent to ROT			
0 ROLL is a null operation			
ROT	16b1 16b2 16b3 -- 16b2 16b3 16b1	79	"rote"
The top three stack entries are rotated, bringing the deepest to the top.			
SAVE-BUFFERS	--	M,79	"save-buffers"
The contents of all block buffers marked as UPDATeEd are written to their corresponding mass storage blocks. All buffers are marked as no longer being modified, but may remain assigned.			
SIGN	n --	83	
If n is negative, an ASCII "-" (minus sign) is appended to the pictured numeric output string. Typically used between <# and #> .			
SPACE	--	M,79	
Displays an ASCII space.			
SPACES	+n --	M,79	
Displays +n ASCII spaces. Nothing is displayed if +n is zero.			

SPAN	-- addr	U,83	
The address of a variable containing the count of characters actually received and stored by the last execution of EXPECT . See: EXPECT			
STATE	-- addr	U,79	
The address of a variable containing the compilation state. A non-zero content indicates compilation is occurring, but the value itself is system dependent. A Standard Program may not modify this variable.			
SWAP	16b1 16b2 -- 16b2 16b1	79	
The top two stack entries are exchanged.			
THEN	--	C,I,79	
	sys -- (compiling)		
Used in the form:			
	flag IF ... ELSE ... THEN		
	or		
	flag IF ... THEN		
THEN is the point where execution continues after ELSE , or IF when no ELSE is present. sys is balanced with its corresponding IF or ELSE . See: IF ELSE			
TIB	-- addr	83	"t-i-b"
The address of the text input buffer. This buffer is used to hold characters when the input stream is coming from the current input device. The minimum capacity of TIB is 80 characters.			
TYPE	addr +n --	M,79	
+n characters are displayed from memory beginning with the character at addr and continuing through consecutive addresses. Nothing is displayed if +n is zero. See: "9.5.4 TYPE"			
U.	u --	M,79	"u-dot"
u is displayed as an unsigned number in a free-field format.			
U<	u1 u2 -- flag	83	"u-less-than"
flag is true is u1 is less than u2.			
UM*	u1 u2 -- ud	83	"u-m-times"
ud is the unsigned product of u1 times u2. All values and arithmetic are unsigned.			
UM/MOD	ud u1 -- u2 u3	83	"u-m-divide-mod"
u2 is the remainder and u3 is the floor of the quotient after dividing ud by the divisor u1. All values and arithmetic are unsigned. An error condition results if the divisor is zero or if the quotient lies outside the range {0..65.535}. See: "floor. arithmetic"			

[illegible]

BEGIN ... flag UNTIL

UPDATE -- 79

The currently valid block buffer is marked as modified. Blocks marked as modified will subsequently be automatically transferred to mass storage should its memory buffer be needed for storage of a different block or upon execution of FLUSH or SAVE-BUFFERS .

VARIABLE <name>

VOCABULARY <name>

```

WHILE      flag --                                C,I,79
           sys1 -- sys2      (compiling)

```

BEGIN ... flag WHILE ... REPEAT

Selects conditional execution based on flag. When flag is true, execution continues to just after the WHILE through to the REPEAT which then continues execution back to just after the BEGIN . When flag is false, execution continues to just after the REPEAT , exiting the control structure. sys1 is balanced with its corresponding BEGIN . sys2 is balanced with its corresponding REPEAT . See: BEGIN

12. REQUIRED WORD SET

WORD char -- addr M,83
 Generates a counted string by non-destructively accepting characters from the input stream until the delimiting character char is encountered or the input stream is exhausted. Leading delimiters are ignored. The entire character string is stored in memory beginning at addr as a sequence of bytes. The string is followed by a blank which is not included in the count. The first byte of the string is the number of characters (0..255). If the string is longer than 255 characters, the count is unspecified. If the input stream is already exhausted as WORD is called, then a zero length character string will result.

If the delimiter is not found the value of >IN is the size of the input stream. If the delimiter is found >IN is adjusted to indicate the offset to the character following the delimiter. #TIB is unmodified.

The counted string returned by WORD may reside in the "free" dictionary area at HERE or above. Note that the text interpreter may also use this area. See: "input stream"

XOR 16b1 16b2 -- 16b3 79 "x-or"
 16b3 is the bit-by-bit exclusive-or of 16b1 with 16b2.

[-- I,79 "left-bracket"
 -- (compiling)
 Sets interpret state. The text from the input stream is subsequently interpreted. For typical usage see LITERAL . See:]

['] -- addr C,I,M,83 "bracket-tick"
 -- (compiling)
 Used in the form:
 ['] <name>
 Compiles the compilation address addr of <name> as a literal. When the colon definition is later executed addr is left on the stack. An error condition exists if <name> is not found in the currently active search order. See: LITERAL

[COMPILE] -- C,I,M,79 "bracket-compile"
 -- (compiling)
 Used in the form:
 [COMPILE] <name>
 Forces compilation of the following word <name>. This allows compilation of an immediate word when it would otherwise have been executed.

] -- 79 "right-bracket"
 Sets compilation state. The text from the input stream is subsequently compiled. For typical usage see LITERAL . See: [

13. DOUBLE NUMBER EXTENSION WORD SET

13. DOUBLE NUMBER EXTENSION WORD SET

13.1 The Double Number Extension Word Set Layers

Nucleus layer

2! 2@ 2DROP 2DUP 2OVER 2ROT 2SWAP D+ D- D0= D2/
D< D= DABS DMAX DMIN DNEGATE DU<

Device Layer

none

Interpreter layer

D. D.R

Compiler layer

2CONSTANT 2VARIABLE

13. DOUBLE NUMBER EXTENSION WORD SET

13.2 The Double Number Extension Word Set Glossary

2I	32b addr -- 32b is stored at addr. See: "number"	79	"two-store"
2@	addr -- 32b 32b is the value at addr. See: "number"	79	"two-fetch"
2CONSTANT	32b -- A defining word executed in the form: 32b 2CONSTANT <name> Creates a dictionary entry for <name> so that when <name> is later executed, 32b will be left on the stack.	M,83	"two-constant"
2DROP	32b -- 32b is removed from the stack.	79	"two-drop"
2DUP	32b -- 32b 32b Duplicate 32b.	79	"two-dupe"
2OVER	32b1 32b2 -- 32b1 32b2 32b3 32b3 is a copy of 32b1.	79	"two-over"
2ROT	32b1 32b2 32b3 -- 32b2 32b3 32b1 The top three double numbers on the stack are rotated, bringing the third double number to the top of the stack.	79	"two-rote"
2SWAP	32b1 32b2 -- 32b2 32b1 The top two double numbers are exchanged.	79	"two-swap"
2VARIABLE	-- A defining word executed in the form: 2VARIABLE <name> A dictionary entry for <name> is created and four bytes are ALLOTTed in its parameter field. This parameter field is to be used for contents of the variable. The application is responsible for initializing the contents of the variable which it creates. When <name> is later executed, the address of its parameter field is placed on the stack. See: VARIABLE	M,79	"two-variable"
D+	wd1 wd2 -- wd3 See the complete definition in the Required Word Set.	79	
D-	wd1 wd2 -- wd3 wd3 is the result of subtracting wd2 from wd1.	79	"d-minus"
D.	d -- The absolute value of d is displayed in a free field format. A leading negative sign is displayed if d is negative.	M,79	"d-dot"

13. DOUBLE NUMBER EXTENSION WORD SET

D.R	d +n --	M,83	"d-dot-r"
	d is converted using the value of BASE and then displayed right aligned in a field +n characters wide. A leading minus sign is displayed if d is negative. If the number of characters required to display d is greater than +n, an error condition exists. See: "number conversion"		
D0=	wd -- flag	83	"d-zero-equals"
	flag is true if wd is zero.		
D2/	d1 -- d2	83	"d-two-divide"
	d2 is the result of d1 arithmetically shifted right one bit. The sign is included in the shift and remains unchanged.		
D<	d1 d2 -- flag	83	
	See the complete definition in the Required Word Set.		
D=	wd1 wd2 -- flag	83	"d-equal"
	flag is true if wd1 equals wd2.		
DABS	d -- ud	79	"d-absolute"
	ud is the absolute value of d. If d is -2,147,483,648 then ud is the same value. See: "arithmetic, two's complement"		
DMAX	d1 d2 -- d3	79	"d-max"
	d3 is the greater of d1 and d2.		
DMIN	d1 d2 -- d3	79	"d-min"
	d3 is the lesser of d1 and d2.		
DNEGATE	d1 -- d2	79	
	See the complete definition in the Required Word Set.		
DU<	ud1 ud2 -- flag	83	"d-u-less"
	flag is true if ud1 is less than ud2. Both numbers are unsigned.		

14. ASSEMBLER EXTENSION WORD SET**14. ASSEMBLER EXTENSION WORD SET****14.1 The Assembler Extension Word Set Layers**

Nucleus layer

none

Device layer

none

Interpreter layer

ASSEMBLER

Compiler layer

;CODE CODE END-CODE

14.2 Assembler Extension Word Set Usage

Because of the system dependent nature of machine language programming, a Standard Program cannot use CODE or ;CODE .

14. ASSEMBLER EXTENSION WORD SET

14.3 The Assembler Extension Word Set Glossary

;CODE -- C,I,79 "semi-colon-code"
 sys1 -- sys2 (compiling)

Used in the form:

: <name> ... <create> ... ;CODE ... END-CODE

Stops compilation, terminates the defining word <name> and executes ASSEMBLER. When <name> is executed in the form:

<name> <name>

to define the new <name>, the execution address of <name> will contain the address of the code sequence following the ;CODE in <name>. Execution of any <name> will cause this machine code sequence to be executed. sys1 is balanced with its corresponding : . sys2 is balanced with its corresponding END-CODE . See: CODE DOES

ASSEMBLER -- 83
 Execution replaces the first vocabulary in the search order with the ASSEMBLER vocabulary. See: VOCABULARY

CODE -- sys M,83
 A defining word executed in the form:
 CODE <name> ... END-CODE
 Creates a dictionary entry for <name> to be defined by a following sequence of assembly language words. Words thus defined are called code definitions. This newly created word definition for <name> cannot be found in the dictionary until the corresponding END-CODE is successfully processed (see: END-CODE). Executes ASSEMBLER . sys is balanced with its corresponding END-CODE .

END-CODE sys -- 79 "end-code"
 Terminates a code definition and allows the <name> of the corresponding code definition to be found in the dictionary. sys is balanced with its corresponding CODE or ;CODE . See: CODE

15. THE SYSTEM EXTENSION WORD SET

15. THE SYSTEM EXTENSION WORD SET

15.1 The System Extension Word Set Layers

Nucleus layer

BRANCH ?BRANCH

Device layer

none

Interpreter layer

CONTEXT CURRENT

Compiler layer

<MARK <RESOLVE >MARK >RESOLVE

15.2 System Extension Word Set Usage

After BRANCH or ?BRANCH is compiled, >MARK or <RESOLVE is executed. The addr left by >MARK is passed to >RESOLVE. The addr left by <MARK is passed to <RESOLVE. For example:

```
: IF    COMPILE ?BRANCH >MARK    ; IMMEDIATE
: THEN  >RESOLVE    ; IMMEDIATE
```

15. THE SYSTEM EXTENSION WORD SET

15.3 The System Extension Word Set Glossary

<MARK	-- addr	C,83	"backward-mark"
Used at the destination of a backward branch. addr is typically only used by <RESOLVE to compile a branch address.			
<RESOLVE	addr --	C,83	"backward-resolve"
Used at the source of a backward branch after either BRANCH or ?BRANCH . Compiles a branch address using addr as the destination address.			
>MARK	-- addr	C,83	"forward-mark"
Used at the source of a forward branch. Typically used after either BRANCH or ?BRANCH . Compiles space in the dictionary for a branch address which will later be resolved by >RESOLVE .			
>RESOLVE	addr --	C,83	"forward-resolve"
Used at the destination of a forward branch. Calculates the branch address (to the current location in the dictionary) using addr and places this branch address into the space left by >MARK .			
?BRANCH	flag --	C,83	"question-branch"
When used in the form: COMPILE ?BRANCH a conditional branch operation is compiled. See BRANCH for further details. When executed, if flag is false the branch is performed as with BRANCH . When flag is true execution continues at the compilation address immediately following the branch address.			
BRANCH	--	C,83	
When used in the form: COMPILE BRANCH an unconditional branch operation is compiled. A branch address must be compiled immediately following this compilation address. The branch address is typically generated by following BRANCH with <RESOLVE or >MARK .			
CONTEXT	-- addr	U,79	
The address of a variable which determines the dictionary search order.			
CURRENT	-- addr	U,79	
The address of a variable specifying the vocabulary in which new word definitions are appended.			

16. CONTROLLED REFERENCE WORDS

16. CONTROLLED REFERENCE WORDS

END flag -- C,I,79
 sys -- (compiling)
 A synonym for UNTIL .

ERASE addr u -- 79
 u bytes of memory beginning at addr are set to zero. No action is taken
 if u is zero.

HEX -- 79
 Set the numeric input-output conversion base to sixteen.

INTERPRET -- M,83
 Begin text interpretation at the character indexed by the contents of >IN
 relative to the block number contained in BLK , continuing until the
 input stream is exhausted. If BLK contains zero, interpret characters
 from the text input buffer. See: "input stream"

K -- w C,83
 w is a copy of the index of the second outer loop. May only be used
 within a nested DO-LOOP or DO-+LOOP in the form, for example:
 DO ... DO ... DO ... K ... LOOP ... +LOOP ... LOOP

LIST u -- M,79
 The contents of screen u are displayed. SCR is set to u. See: BLOCK

OCTAL -- 83
 Set the numeric input-output conversion base to eight.

OFFSET -- addr U,83
 The address of a variable that contains the offset added to the block
 number on the stack by BLOCK or BUFFER to determine the actual physical
 block number.

QUERY -- M,83
 Characters are received and transferred into the memory area addressed by
 TIB . The transfer terminates when either a "return" is received or the
 number of characters transferred reaches the size of the area addressed
 by TIB . The values of >IN and BLK are set to zero and the value of #TIB
 is set to the value of SPAN . WORD may be used to accept text from this
 buffer. See: EXPECT "input stream"

RECURSE -- C,I,83
 -- (compiling)
 Compile the compilation address of the definition being compiled to cause
 the definition to later be executed recursively.

SCR -- addr U,79 "s-c-r"
 The address of a variable containing the number of the screen most
 recently LISTed.

16. CONTROLLED REFERENCE WORDS

SP@ -- addr 79 "s-p-fetch"
addr is the address of the top of the stack just before SP@ was executed.

THRU u1 u2 -- M,83
Load consecutively the blocks from u1 through u2.

U.R u +n -- M,83 "u-dot-r"
u is converted using the value of BASE and then displayed as an unsigned
number right aligned in a field +n characters wide. If the number of
characters required to display u is greater than +n, an error condition
exists. See: "number conversion"

A. STANDARDS TEAM MEMBERSHIP

APPENDIX A. STANDARDS TEAM MEMBERSHIP

A.1 Standards Team Membership: Members

The following is a list in alphabetical order of the people who are FORTH Standards Team Members. These names are provided to indicate the texture and make-up of the team itself. Where appropriate, the official capacity of individuals is also indicated.

Paul Bartholdi, Sauverny, Switzerland	
Robert Berkey, Palo Alto, California USA	Treasurer
David Boulton, Redwood City, California USA	
John Bumgarner, Morgan Hill, California USA	
Don Colburn, Rockville, Maryland USA	
James T. Currie, Jr., Blacksburg, Virginia USA	
Thomas B. Dowling, Lowell, Massachusetts USA	
William S. Emery, Malibu, California USA	
Lawrence P. Forsley, Rochester, New York USA	
Kim R. Harris, Palo Alto, California USA	Referee
John S. James, Los Gatos, California USA	
Guy M. Kelly, La Jolla, California USA	Chairperson
Thea Martin, Rochester, New York USA	
Michael McNeil, Scotts Valley, California USA	
Robert E. Patten, Modesto, California USA	
Michael Perry, Berkeley, California USA	
David C. Petty, Cambridge, Massachusetts USA	
William F. Ragsdale, Hayward, California USA	
Elizabeth D. Rather, Hermosa Beach, California USA	
Dean Sanderson, Hermosa Beach, California USA	Referee
Klaus Schleisiek, Hamburg, W-Germany	
George W. Shaw II, Hayward, California USA	Referee
Robert L. Smith, Palo Alto, California USA	Secretary
Michael K. Starling, Elkview, West Virginia USA	
John K. Stevenson, Portland, Oregon USA	
Glenn S. Tenney, San Mateo, California USA	Referee

A. STANDARDS TEAM MEMBERSHIP**A.2 FORTH Standards Team Sponsors**

The following is a list in alphabetical order of individuals and organizations who have contributed funds and other assistance to aid the work of the FST and deserve recognition for their involvement. FST sponsors have no duties or responsibilities in the FST, but they receive copies of proposals and comments considered at a formal meeting, and drafts and adopted standards prepared as a result of that meeting.

Creative Solutions Inc., 4881 Randolph Rd., Rockville, MD 20852 USA
Fantasia Systems Inc., 1859 Alameda de las Pulgas, Belmont, CA 94002 USA
FORTH, Inc., 2389 Pacific Coast Highway, Hermosa Beach, CA 90254 USA
FORTH Interest Group Inc., P.O. Box 1185, San Carlos, CA 94070 USA
Forthright Enterprises, P.O. Box 58911, Palo Alto, CA 94303 USA
Glen Haydon Enterprises, Box 439 Rt. 2, La Honda, CA 94020 USA
John K. Gotwals, W. Lafayette, IN USA
John D. Hall, Oakland, CA USA
Hartronix, Inc., 1281 N. Stadem, Tempe, AZ 85281 USA
Hewlett-Packard Corvallis Div., 1000 NE Circle Blvd., Corvallis, OR 97330 USA
Information Unlimited Software, Inc., 2481 Marinship, Sausalito, CA 94965 USA
Inner Access Corporation, P. O. Box 888, Belmont, CA 94002 USA
Laboratory Microsystems, 4147 Beethoven St., Los Angeles, CA 90066 USA
Henry H. Laxen, 1259 Cornell Avenue, Berkeley, CA 94706 USA
Laxen & Harris, Inc.
George B. Lyons, 288 Henderson Street, Jersey City, NJ 07302 USA
C. Kevin McCabe, Chicago, IL USA
MicroMotion, 12877 Wilshire Boulevard #586, Los Angeles, CA 90025 USA
Bruce R. Montague, Monterey, CA USA
Mountain View Press, P.O. Box 4659, Mountain View, CA 94040 USA
Michael A. Perry, Berkeley, CA USA

A. STANDARDS TEAM MEMBERSHIP

Robert Berkey Services, 2334 Dumbarton Ave., Palo Alto, CA 94303 USA

Royal Greenwich Observatory, Herstmonsioux Castle, Eastbourne, England

Shaw Laboratories, Ltd., 24301 Southland Drive #216, Hayward, CA 94545 USA

Sygnatron Protection Systems, Inc., 2103 Greenspring, Timonium, MD 21093 USA

Telelogic Inc., 196 Broadway, Cambridge, MA 02139 USA

UNISOFT, P.O. Box 2644, New Carrollton, MD 20784 USA

B. UNCONTROLLED REFERENCE WORDS

APPENDIX B. UNCONTROLLED REFERENCE WORDS

The Uncontrolled Reference Word Set contains glossary definitions which are included for public reference of words that have past or present usage and/or are candidates for future standardization. No recommendation is made that these words be included in a system.

No restrictions are placed on the definition or usage of uncontrolled words. However, use of these names for procedures differing from the given definitions is discouraged.

1BITS 16b1 addr 16b2 -- "store-bits"
Store the value of 16b1 masked by 16b2 into the equivalent masked part of the contents of addr, without affecting bits outside the mask.

****** n1 n2 -- n3 "power"
n3 is the value of n1 to the power n2.

+BLOCK w -- u "plus-block"
u is the sum of w plus the number of the block being interpreted.

-' -- addr false M "dash-tick"
 -- true
Used in the form:
- ' <name>
Leave the parameter field of <name> beneath zero (false) if <name> can be found in the search order; leave only true if not found.

-MATCH addr1 +n1 addr2 +n2 -- addr3 flag "dash-match"
Attempt to find the +n2-length text string beginning at addr2 somewhere in the +n1-length text string beginning at addr1. Return the last+1 address addr3 of the match point and a flag which is zero if a match exists.

-TEXT addr1 +n1 addr2 -- n2 "dash-text"
Compare two strings over the length +n1 beginning at addr1 and addr2. Return zero if the strings are equal. If unequal, return n2, the difference between the last characters compared: addr1(i) - addr2(i).

B. UNCONTROLLED REFERENCE WORDS

```

/LOOP          +n --          C,I          "up-loop"
              sys --          (compiling)

A do-loop terminating word. The loop index is incremented by the
positive value +n. If the unsigned magnitude of the resultant index is
greater than the limit, then the loop is terminated, otherwise execution
returns to the corresponding DO . The comparison is unsigned magnitude.
sys is balanced with its corresponding DO . See: DO

1+!           addr --          "one-plus-store"
Add one to the 16-bit contents at addr.

1-!           addr --          "one-minus-store"
Subtract one from the 16-bit contents at addr.

;;            -- addr          C,I          "semi-colon-colon"
Used to specify a new defining word:
: <namex> ... ;; ... ;
<namex> <name>
When <namex> is executed, it creates an entry for the new word <name>.
Later execution of <name> will execute the sequence of words between ;;
and ; , with the address of the first (if any) parameters associated with
<name> on the stack.

;S            --              Interpret only  "semi-s"
Stop interpretation of a block.

<>            w1 w2 -- flag    "not-equal"
flag is true if w1 is not equal to w2.

<BUILDS      --              C,M          "builds"
Used in conjunction with DOES> in defining words, in the form:
: <namex> ... <BUILDS ... DOES> ... ;
and then <namex> <name>

When <namex> executes, <BUILDS creates a dictionary entry for the new
<name>. The sequence of words between <BUILDS and DOES> established a
parameter field for <name>. When <name> is later executed, the
sequence of words following DOES> will be executed, with the parameter
field address of <name> on the data stack.

<CMOVE       addr1 addr2 u --    "reverse-c-move"
A synonym for CMOVE> .

><            16b1 -- 16b2      "byte-swap"
Swap the high and low bytes within 16b1.

>MOVE<       addr1 addr2 u --    "byte-swap-move"
Move u bytes beginning at addr1 to the memory beginning at addr2. During
this move, the order of each byte pair is reversed.

@BITS        addr 16b1 -- 16b2   "fetch-bits"
Return the 16-bits at addr masked by 16b1.

```

```

AGAIN          -- sys --      (compiling)          C,I
Effect an unconditional jump back to the start of a BEGIN-AGAIN loop.
sys is balanced with its corresponding BEGIN . See: BEGIN

ASCII          -- char          I,M          "as-key"
              --      (compiling)

Used in the form:
      ASCII ccc
where the delimiter of ccc is a space. char is the ASCII character
value of the first character in ccc. If interpreting, char is left on
the stack. If compiling, compile char as a literal so that when the
colon definition is later executed, char is left on the stack.

ASHIFT          16b1 n -- 16b2          "a-shift"
Shift the value 16b1 arithmetically n bits left if n is positive,
shifting zeros into the least-significant bit positions. If n is
negative, 16b1 is shifted right; the sign is included in the shift and
remains unchanged.

B/BUF          -- 1024          "bytes-per-buffer"
A constant leaving 1024, the number of bytes per block buffer.

BELL          --      M
Activate a terminal bell or noise-maker as appropriate to the device in
use.

CHAIN          --      M
Used in the form:
      CHAIN <name>
Connect the CURRENT vocabulary to all definitions that might be entered
into the vocabulary <name> in the future. The CURRENT vocabulary may
not be FORTH or ASSEMBLER . Any given vocabulary may only be chained
once, but may be the object of any number of chainings. For example,
every user-defined vocabulary may include the sequence:
      CHAIN FORTH

CONTINUED          u --      M
Continue interpretation at block u.

CUR          -- addr
A variable pointing to the physical record number before which the tape
is currently positioned. REWIND sets CUR=1.

DBLOCK          ud -- addr          M          "d-block"
Identical to BLOCK but with a 32-bit block unsigned number.

DPL          -- addr          U          "d-p-1"
A variable containing the number of places after the fractional point for
input conversion.

```

B. UNCONTROLLED REFERENCE WORDS

```

FLD          -- addr          U          "f-l-d"
A variable pointing to the field length reserved for a number during
output conversion.

H.           u --            M          "h-dot"
Output u as a hexadecimal integer with one trailing blank. The current
base is unchanged.

I'           -- w            C          "i-prime"
Used within a colon definition executed only from within a do-loop to
return the corresponding loop index.

IFEND                               Interpret only  "if-end"
Terminate a conditional interpretation sequence begun by IFTRUE .

IFTRUE      flag --          Interpret only  "if-true"
Begin an
    IFTRUE ... OTHERWISE ... IFEND
conditional sequence. These conditional words operate like
    IF ... ELSE ... THEN
except that they cannot be nested, and are to be used only during
interpretation. In conjunction with the words [ and ] they may be used
within a colon definition to control compilation, although they are not
to be compiled.

INDEX       u1 u2 --        M
Print the first line of each screen over the range {u1..u2}. This
displays the first line of each screen of source text, which
conventionally contains a title.

LAST        -- addr          U
A variable containing the address of the beginning of the last dictionary
entry made, which may not yet be a complete or valid entry.

LINE        +n -- addr      M
addr is the address of the beginning of line +n for the screen whose
number is contained in SCR . The range of +n is {0..15}.

LINELOAD    +n u --        M          "line-load"
Begin interpretation at line +n of screen u.

LOADS       u --            M
A defining word executed in the form:
    u LOADS <name>
When <name> is subsequently executed, block u will be loaded.

MAP$        -- addr          "map-zero"
A variable pointing to the first location in the tape map.

MASK        n -- 16b
16b is a mask of n most-significant bits if n is positive, or n least-
significant bits if n is negative.

```

B. UNCONTROLLED REFERENCE WORDS

- MOVE** `addr1 addr2 u --`
 The u bytes at address addr1 are moved to address addr2. The data are moved such that the u bytes remaining at address addr2 are the same data as was originally at address addr1. If u is zero nothing is moved.
- MS** `+n --` M "n-s"
 Delay for approximately +n milliseconds.
- NAND** `16b1 16b2 -- 16b3`
 16b3 is the one's complement of the logical AND of 16b1 with 16b2.
- NOR** `16b1 16b2 -- 16b3`
 16b3 is the one's complement of the logical OR of 16b1 with 16b2.
- NUMBER** `addr -- d`
 Convert the count and character string at addr, to a signed 32-bit integer, using the value of BASE . If numeric conversion is not possible, an error condition exists. The string may contain a preceding minus sign.
- O.** `u --` M "o-dot"
 Print u in octal format with one trailing blank. The value in BASE is unaffected.
- OTHERWISE** `--` Interpret only
 An interpreter-level conditional word. See: IFTRUE
- PAGE** `--` M
 Clear the terminal screen or perform a form-feed action suitable to the output device currently active.
- READ-MAP** `--` M "read-map"
 Read to the next file mark on tape constructing a correspondence table in memory (the map) relating physical block position to logical block number. The tape should normally be rewound to its load point before executing READ-MAP .
- REMEMBER** `--` M
 A defining word executed in the form:
 REMEMBER `<name>`
 Defines a word which, when executed, will cause `<name>` and all subsequently defined words to be deleted from the dictionary. `<name>` may be compiled into and executed from a colon definition. The sequence
 DISCARD REMEMBER DISCARD
 provides a standardized preface to any group of transient word definitions.
- REWIND** `--` M
 Rewind the tape to its load point, setting CUR equal to one.
- ROTATE** `16b1 n -- 16b2`
 Rotate 16b1 left n bits if n is positive, right n bits if n is negative. Bits shifted out of one end of the cell are shifted back in at the

B. UNCONTROLLED REFERENCE WORDS

opposite end.

SD	-- addr	U	"s-zero"
A variable containing the address of the bottom of the stack.			

```

SET          16b addr --                                M
A defining word executed in the form:
    16b addr SET <name>
Defines a word <name> which, when executed, will cause the value 16b to
be stored at addr.

```

SHIFT 16b1 n -- 16b2
Logical shift 16b1 left n bits if n is positive, right n bits if n is
negative. Zeros are shifted into vacated bit positions.

```
TEXT      char --                                M
Accept characters from the input stream, as for WORD , into PAD ,
blank-filling the remainder of PAD to 84 characters.
```

```

USER          +n --                                M
A defining word executed in the form:
    +n USER <name>
which creates a user variable <name>. +n is the offset within the user
area where the value for <name> is stored. Execution of <name>
leaves its absolute user area storage address.

```

```

WORDS      --                               M
List the word names in the first vocabulary of the currently active
search order.

```

```

\LOOP      +n --                      C,I      "down-loop"
           sys --      (compiling)
A do-loop terminating word. The loop index is decremented by the
positive value +n. If the unsigned magnitude of the resultant index is
less than or equal to the limit, then the loop is terminated, otherwise
execution returns to the corresponding DO . The comparison is unsigned.
sys is balanced with its corresponding DO . See: DO

```

C. EXPERIMENTAL PROPOSALS**APPENDIX C. EXPERIMENTAL PROPOSALS**

Since FORTH is an extensible language and subject to evolution, the Standard contains a section describing experimental proposals. FORTH users are encouraged to study, implement, and try these proposals to aid in the analysis of and the decision for or against future adoption into the Standard. Readers are cautioned that these proposals contain opinions and conclusions of the authors of the proposals and that these proposals may contain non-standard source code.

C. EXPERIMENTAL PROPOSALS

SEARCH ORDER SPECIFICATION AND CONTROL

WILLIAM F. RAGSDALE

1 INTRODUCTION

The method of selecting the order in which the dictionary is searched has grown from unchained vocabularies to the present use of chained vocabularies. Many techniques are in use for specification of the sequence in which multiple vocabularies may be searched. In order to offer generality and yet get precision in specification, this proposal is offered.

2 DESCRIPTION

The following functions are required:

1. Two search orders exist. CONTEXT is the group of vocabularies searched during interpretation of text from the input stream. CURRENT is the single vocabulary into which new definitions are compiled, and from which FORGET operates.
2. Empty CONTEXT to a minimum number of system words. These are just the words to further specify the search order.
3. Add individual vocabularies into CONTEXT. The most recently added is searched first.
4. Specify which single vocabulary will become CURRENT.

The following optional functions aid the user:

1. Display the word names of the first vocabulary in the CONTEXT search order.
2. Display the vocabulary names comprising CURRENT and CONTEXT search orders.

3 ADVANTAGES

Use over the past year has demonstrated that the proposed methods may emulate the vocabulary selection of all other systems. The order is explicit by execution, may be interpreted and compiled, and is obvious from the declaration. The search order is specified at run-time rather than the time a new vocabulary is created.

C. EXPERIMENTAL PROPOSALS

4 DISADVANTAGES

By migrating to a common structure, vendors give up one point at which they may claim their product is better than others. Another drawback is that the number of CONTEXT vocabularies is fixed; older methods had an indefinite 'tree' structure. In practice, the branching of such a structure was very rarely greater than four.

Forth words operate in a context sensitive environment, as word names may be redefined and have different definitions in different vocabularies. This proposal compounds the problem. By displaying the search order names, the user at least can readily verify the search order.

5 IMPACT

The text of the Forth 83 Standard has been carefully chosen for consistency and generality. However, no specification on how the search order is developed by the user is given. This omission is unavoidable, due to the diversity of contemporary practice. This proposal is intended to complete the Forth 83 requirements in a fashion that exceeds all other methods.

Previously standardized words continue in their use: VOCABULARY, FORTH, DEFINITIONS, and FORGET. However, this proposal assumes that vocabulary names are not IMMEDIATE .

6 DEFINITIONS

Search order:

The sequence in which vocabularies are selected when locating a word by name in the dictionary. Consists of one transient and up to three resident vocabularies.

Transient order:

Execution of any vocabulary makes it the first vocabulary searched, replacing the previously selected transient vocabulary.

Resident order:

After searching the transient order, up to three additional vocabularies may be searched. The application program controls this selection.

7 GLOSSARY

ONLY

ONLY

Select just the ONLY vocabulary as both the transient vocabulary and resident vocabulary in the search order

C. EXPERIMENTAL PROPOSALS

FORTH	---	ONLY
The name of the primary vocabulary. Execution makes FORTH the transient vocabulary, the first in the search order, and thus replaces the previous transient vocabulary.		
ALSO	---	ONLY
The transient vocabulary becomes the first vocabulary in the resident portion of the search order. Up to the last two resident vocabularies will also be reserved, in order, forming the resident search order.		
ORDER	---	ONLY
Display the vocabulary names forming the search order in their present search order sequence. Then show the vocabulary into which new definitions will be placed.		
WORDS	---	ONLY
Display the word names in the transient vocabulary, starting with the most recent definition.		
FORGET	---	ONLY
Used in the form: FORGET <name> Delete from the dictionary <name> and all words added to the dictionary after <name> regardless of the vocabulary. Failure to find <name> is an error condition. An error condition also exists upon implicitly forgetting a vocabulary (due to its definition after <name>).		
DEFINITIONS	---	ONLY
Select the transient vocabulary as the current vocabulary into which subsequent definitions will be added.		
SEAL	---	ONLY
Delete all occurrences of ONLY from the search order. The effect is that only specified application vocabularies will be searched.		

C. EXPERIMENTAL PROPOSALS

8 TYPICAL SOURCE CODE

```

8 ( ALSO ONLY                                     82jun12 WFR )
1 ( note the systems -FIND searches 1 to 5 vocabs in CONTEXT )
2 VOCABULARY ONLY ONLY DEFINITIONS
3 : ALSO ( slide transient into resident )
4   CONTEXT DUP 2+ 6 CMOVE> ;
5
6   HERE 2+ ] ( alter run time from usual vocabulary )
7   DOES> CONTEXT 8 ERASE DUP CONTEXT ! CONTEXT 8 + !
8   ALSO EXIT [
9   ' ONLY CFA ! ( Patch into ONLY; make NULL word )
10 CREATE X ' EXIT >BODY X ! 41088 ' X NFA ! IMMEDIATE
11 : FORTH FORTH ;
12 : DEFINITIONS DEFINITIONS ; : FORGET FORGET ;
13 : VOCABULARY VOCABULARY ; : ONLY ONLY ;
14 : WORDS WORDS ;
15

```

```

8 ( ORDER                                     82jun12 WFR )
1 : ORDER ( show the search order )
2   10 SPACES CONTEXT 10 OVER + SWAP
3   DO I @ ?DUP 0= ?LEAVE ID. 2 +LOOP
4   10 SPACES CURRENT @ ID. ;
5
6 ONLY FORTH ALSO DEFINITIONS
7
8
9
10
11
12
13
14
15

```

9 EXAMPLES OF USE

ONLY	reduce search order to minimum
FORTH	search FORTH then ONLY
ALSO EDITOR	search EDITOR, FORTH then ONLY
DEFINITIONS	new definitions will be added into the EDITOR

The same sequence would be compiled:

```
: SETUP ONLY FORTH ALSO EDITOR DEFINITIONS ;
```

C. EXPERIMENTAL PROPOSALS

10 REFERENCES

W. F. Ragsdale, The 'ONLY' Concept for Vocabularies, Proceedings of the 1982 FORML Conference, pub. Forth Interest Group.

W. F. Ragsdale, fig-FORTH Installation Manual, Forth Interest Group.

C. EXPERIMENTAL PROPOSALS

DEFINITION FIELD ADDRESS CONVERSION OPERATORS

by

Kim R. Harris

A. INTRODUCTION

The standard provides a transportable way to obtain the compilation address of a definition in the dictionary of a FORTH system (cf., FIND and '). It also provides an operator to convert a compilation address to its corresponding parameter field address. However, the standard does not provide a transportable way to convert either of these addresses to the other fields of a definition. Since various FORTH implementations have different dictionary structures, a standard set of conversion operators would increase transportability and readability.

A set of words is proposed which allows the conversion of any definition field address to any other.

B. GLOSSARY

In the following words, the compilation address is either the source or the destination, so it is not indicated in the names.

```

>BODY      addr1 -- addr2      "to-body"
           addr2 is the parameter field address corresponding to the compilation
           address addr1.

>NAME      addr1 -- addr2      "to-name"
           addr2 is the name field address corresponding to the compilation address
           addr1.

>LINK      addr1 -- addr2      "to-link"
           addr2 is the link field address corresponding to the compilation address
           addr1.

BODY>      addr1 -- addr2      "from-body"
           addr2 is the compilation address corresponding to the parameter field
           address addr1.

NAME>      addr1 -- addr2      "from-name"
           addr2 is the compilation address corresponding to the name field address
           addr1.

LINK>      addr1 -- addr2      "from-link"
           addr2 is the compilation address corresponding to the link field address
           addr1.

```

C. EXPERIMENTAL PROPOSALS

The previous set of words is complete, but may be inefficient for going between two fields when one is not the compilation address. For greater efficiency, additional operators may be defined which name both the source and destination fields.

N>LINK addr1 -- addr2 "name-to-link"
 addr2 is the link field address corresponding to the name field address
 addr1.

L>NAME addr1 -- addr2 "link-to-name"
 addr2 is the name field address corresponding to the link field address
 addr1.

C. DISCUSSION

The previous words provide a complete, consistent, and efficient set of definition field address conversion operations. They can be implemented in a FORTH system which uses any combination of the following options for its dictionary structure:

- Link fields first or second.
- Fixed or variable length name fields.
- Additional fields in the definition structure.

- Heads contiguous or separated from bodies.

- Indirect, direct, subroutine, or token threaded code.

The words are compatible with this standard; their inclusion would not require other changes to be made to the standard.

Disadvantages to including them in the standard include:

- They add 6 to 8 more words to the standard.

- A standard program may not use all of them since it is not allowed to access the name or link fields. However, this does not disqualify them from being in the standard.

- If a definition's head is not in the dictionary, an error condition would exist. In this case, what action should the words take in an implemented system?

The author of this experimental proposal recommends that FORTH system implementors try them and that they be included in the System Word Set of the next FORTH standard.

C. EXPERIMENTAL PROPOSALS

D. SOURCE CODE EXAMPLE

High level source code is shown below for a very simple dictionary structure. This code assumes a FORTH system which uses indirect threaded code, heads contiguous to bodies, and a definition structure of the following format:

Name field, 4 bytes long, fixed length.
Link field, 2 bytes long.
Code field, 2 bytes long.
Parameter field, variable length.

```
: >BODY ( acf -- apf ) 2+ ;  
: BODY> ( apf -- acf ) 2- ;  
: >LINK ( acf -- alf ) 2- ;  
: LINK> ( alf -- acf ) 2+ ;  
: >NAME ( acf -- anf ) 6 - ;  
: NAME> ( anf -- acf ) 6 + ;  
: N>LINK ( anf -- alf ) 4 + ;  
: L>NAME ( alf -- anf ) 4 - ;
```

E. EXAMPLES OF USE

No examples are given because their use should be obvious.

D. CHARTER

APPENDIX D.

CHARTER

of the

FORTH STANDARDS TEAM

1. Purpose and Goals

1.1 Purpose

1.1.1 This Charter establishes and guides a voluntary membership professional organization, the FORTH Standards Team (hereafter referred to as the "FST") and provides a method for its operation.

1.2 Goals

1.2.1 The goal of the FST is the creation, maintenance, and proliferation of a standard (hereafter referred to as the "Standard") for the FORTH computer programming system and for application programs executed by a Standard system. The Standard shall specify requirements and constraints which such computer software must satisfy.

1.2.2 The team shall also develop a method of identification and labeling of FORTH implementations and programs which conform to the Standard.

1.3 Organization

1.3.1 The FST is a voluntary membership organization with no formal status as a legal entity. It operates by consensus of the professional and commercial FORTH community and conducts business by the professional discourse and agreement of its members. It is intended that this Charter be a guide to the operation of the FST subject to reasonable minor digression, rather than being a rigid document under which vested rights are granted.

D. CHARTER

2. METHODS

2.1 Formal Meetings

2.1.1 The FST shall hold periodic formal meetings for discussion and decisions concerning a current or future Standard.

2.1.2 There is no specified frequency for formal meetings. Each meeting shall be at such time and place as was decided at the prior meeting. If a meeting cannot be held as decided, the Chairperson may designate another time and place.

2.1.3 The Chairperson shall send a written notice at least sixty (60) days in advance of each formal meeting to each voting member. A longer notification period is recommended. It is anticipated that the continuing close coordination of the participants, the decision at the prior formal meeting, and publication of a meeting notice in FORTH Dimensions and other trade journals will provide sufficient notice to the FORTH community.

2.1.4 At a formal FST meeting, there shall be general sessions consisting of all attendees. General sessions are for matters that are ready for discussion and decision. All votes concerning the Standard, Charter, or FST procedures must take place during a general session.

2.1.5 Also at formal meetings, subteams will be established to examine groups of proposals and to prepare recommendations for a general session. All meeting attendees may participate in the work and voting of a subteam. Each subteam should elect from its members a coordinator to conduct its meetings and a reporter to record and report its recommendations.

2.1.6 The Chairperson may publish and distribute an agenda at or in advance of a formal meeting. As a guideline, each day of a formal meeting begins with a general session, followed by concurrent subteam meetings followed by another general session.

2.1.7 In view of the voluntary nature of the FST, at least one third of the membership is required to hold a formal meeting. Two thirds of the number of voting members present at the start of each day's first general session shall set the quorum for the remainder of that day.

2.1.8 Between formal meetings, the Chairperson may appoint such informal working groups as is appropriate. Each group may be given a goal and scope to direct its activities. Its conclusions or recommendations must be given to the Chairperson in written form.

2.2 Proposals and Comments

2.2.1 Prior to each formal meeting, the Chairperson may solicit submission of comments and proposals for changes, additions, or deletions to the then-current Standard, the draft Standard or this Charter. A cutoff date may be specified for the submission of such proposals.

D. CHARTER

2.2.2 A considerable amount of information must accompany each proposal to help FST members analyze the proposal. Therefore, submission of proposals and comments shall be according to the format and instructions shown in the "Proposal/Comment Form" included as an Appendix to this Standard. Any proposal not in the appropriate form or received after the cutoff date may not be considered unless the Chairperson deems it to be of sufficient significance.

2.2.3 Unsolicited proposals and comments by volunteers are acknowledged as valuable. Any individual or group may submit proposals and/or comments concerning the Standard or this Charter. These should be sent to the official address of the FST. Properly formatted proposals and comments are preferred. The author or a representative should plan to attend the next formal meeting to emphasize, support, and possibly modify the proposals.

2.2.4 Since the quantity of proposals and comments may exceed the number for which there is time to be voted upon, submission of a proposal does not automatically mean that it will be voted upon at the next formal FST meeting. The Chairperson or some members appointed by the Chairperson or elected by the voting members may screen and organize the received proposals and comments for voting upon at the next formal meeting.

2.2.5 To allow reflection and examination, proposals and comments shall be distributed to FST voting members and sponsors in advance of a formal meeting. Proposals and comments not distributed in advance, including proposals made during a formal meeting, may be considered at the discretion of the Chairperson.

2.3 Draft Standard

After a formal meeting, the referees and officers of the FST shall prepare a draft Standard for review by the then-current FST voting members. The referees and officers shall consolidate proposals accepted by vote during the meeting, resolve any ambiguities or problems, and incorporate these changes with the text of the previous Standard or draft Standard.

2.4 Standard

2.4.1 The referees and officers may, by near unanimous decision (not more than one no vote), declare the draft Standard, as mentioned in the previous paragraph, as being the proposed Standard.

2.4.2 A proposed Standard shall be distributed to all FST voting members for a mail ballot. This ballot shall be based solely on the text of the proposed Standard as distributed.

2.4.3 Each ballot returned shall be signed by the voting member submitting it. An affirmative vote of at least two thirds of the voting members shall adopt the document. Such adoption makes the draft Standard the current, official FST Standard which supersedes all prior Standards.

D. CHARTER

2.5 Charter

2.5.1 At a formal FST meeting, the charter may be amended by a simple majority of voting members present provided that at least one third of all voting members are present; such amendments become effective at the end of the current formal meeting.

2.5.2 At other than a formal FST meeting, the charter may be amended by a simple majority of all voting members, such vote to be taken by signed mail ballots.

3. MEMBERSHIP

3.1 General

Membership in the FST is a privilege, not a right. An invitation for voting membership may be extended to those who the FST feels can contribute to the goals of the Standard and the FST. There are several classes of participation in the efforts of the FST. Membership in each class has no specified term but continues from the time when membership is initiated to the conclusion of the next formal meeting.

3.2 Voting Members

3.2.1 Voting members are individuals who are elected into such membership at the concluding session of a formal FST meeting. Any voting member who resigns between formal meetings shall not be replaced until the membership elections at the conclusion of the next formal meeting. A newly elected voting member gains voting rights only after all voting members have been elected. A significant professional FORTH background is required of voting members.

3.2.2 Each voting member present at a formal meeting shall indicate in writing his or her desire to continue as a voting member. Only these voting members can vote in a general session of a formal meeting on any matters affecting the Standard or the Charter and on the election of all voting members.

3.2.3 Voting members are elected by a simple majority of those voting members present. The number of voting members shall be limited to thirty (30). Individuals eligible to be elected are selected from each of the following ordered categories in order, until the number of voting members reaches the limit.

3.2.3.1 Category 1: current voting members who have actively participated in at least two days of a formal meeting. Voting members are expected to actively participate in subteam meetings and all general sessions.

3.2.3.2 Category 2: current voting members who are not eligible by Category 1, but who have requested in writing that his or her voting membership be maintained.

D. CHARTER

3.2.3.3 Category 3: eligible candidates. Eligible candidates will be presented to the voting members then elected as follows:

3.2.3.3.1 If the number of eligible candidates does not exceed the number of openings for voting membership, each candidate is voted upon and accepted by a simple majority.

3.2.3.3.2 If the number of eligible candidates does exceed the number of openings for voting membership, candidates will be voted upon by ballot whereby each voting member may vote for up to the number of openings remaining. Those candidates receiving the most votes will be elected until there are no more openings for voting membership.

3.3 Candidates

3.3.1 Candidates are individuals who desire to actively participate in and support the FST by becoming voting members.

3.3.2 To be eligible, each Candidate must: declare in writing to the secretary at the first general session of a formal FST meeting that he or she is a Candidate, actively participate in subteam meetings and all general sessions at a formal FST meeting, and have a significant professional background in FORTH. The Chairperson may request information or ask questions of any candidate to determine his or her technical knowledge and experience. Candidates are expected to submit proposals, participate in the discussions of the formal meeting, and contribute to the work and voting of subteams.

3.4 Observers

3.4.1 Observers are individuals who attend a formal meeting but are neither voting members nor candidates. At the discretion of the Chairperson, they may contribute to the discussion at general sessions and to the work of subteams. The number of observers allowed at a formal meeting may be limited by the Chairperson.

3.5 FST Sponsors

3.5.1 FST sponsors are individuals or organizations who contribute funds and other assistance to aid the work of the FST. FST sponsors have no duties or responsibilities in the FST, but they will receive copies of proposals and comments considered at a formal meeting, and drafts and adopted standards prepared as a result of that meeting.

3.5.2 FST sponsorship exists from the end of one formal meeting to the end of the next formal meeting.

3.5.3 Qualification of FST sponsors may be determined by a simple majority vote at a formal FST meeting. If no such qualifications exist, the Chairperson may specify qualifications, including the amount of financial contributions, which will remain in effect until the next formal FST meeting.

D. CHARTER

4. OFFICERS

4.1 General

There shall be four types of elected officers of the FST: the Chairperson, the Secretary, the Treasurer, and one or more Referees. Each officer shall be elected at a formal meeting of the FST and serve until the next formal meeting.

4.2 Vacancies

If any office other than the Chairperson becomes vacant between formal meetings, the Chairperson may appoint a replacement. If the office of the Chairperson becomes vacant between formal meetings, a new Chairperson shall be elected by an informal majority vote of the remaining officers. At any formal meeting, any officer, including the Chairperson, may be replaced by a simple majority vote of the voting members present at that meeting.

4.3 Chairperson

4.3.1 The Chairperson is responsible for governing the general business of the FST. He or she is responsible for implementing the FST's Charter and any other requirements specified by the Standard.

4.3.2 The Chairperson's term of office shall be from the conclusion of the formal meeting at which he or she is elected to the conclusion of the next formal meeting. The election of a Chairperson is held at the concluding general session of a formal meeting after the election of voting members; hence, newly elected voting members may vote for the Chairperson. Only voting members are eligible to be elected Chairperson.

4.3.3 The Chairperson shall conduct each formal meeting. In general, the meetings will follow the current Robert's Rules of Order; however, the Chairperson may determine the specific rules for a formal meeting.

4.3.4 Any matter needing a decision between formal meetings not specified by this Charter shall be decided by the Chairperson.

4.3.5 The Chairperson has duties and responsibilities specified elsewhere in this Charter.

4.4 Secretary

4.4.1 The Secretary is responsible for recording the activities and results of the FST.

4.4.2 The Secretary is elected at the first general session of a formal meeting and serves until a Secretary is elected at the beginning of the next formal meeting.

4.4.3 The Secretary has many responsibilities.

D. CHARTER

4.4.3.1 The secretary is responsible for collecting, maintaining and archiving the official copies of the Standard, the Charter, all other FST documents, correspondence, and lists of the FST members of each class.

4.4.3.2 During a formal meeting, the Secretary is responsible for:

(a) Keeping the minutes of the general sessions, including all votes taken. For votes affecting the Standard or Charter, he or she shall: record the number of voting members present, determine if a quorum is present, determine the number of affirmative votes required for the vote to pass, the number of voting members voting in the affirmative and negative, and the result of the vote.

(b) Recording and verifying the attendance and membership class of each attendee.

(c) Recording the recommendations of subteams.

4.4.3.3 The Secretary is also responsible for collecting, archiving, and distributing proposals before a formal meeting. He or she is also responsible for incorporating proposals accepted during a formal meeting into the Standard or Charter. Other officers aid the Secretary in these duties.

4.5 Treasurer

4.5.1 The Treasurer is responsible for managing the financial business of the FST. He or she is responsible for maintaining accurate and current financial records and for accepting and dispersing funds for official FST activities.

4.5.2 The Treasurer's term of office shall be from the conclusion of the formal meeting at which he or she is elected to the conclusion of the next formal meeting. The election of a Treasurer is held just after the election of the Chairperson. Only voting members are eligible to be elected Treasurer.

4.6 Referees

4.6.1 At the conclusion of a formal meeting there may be additional technical work required to prepare a draft Standard or Charter. This work shall be performed by the officers of the FST, including a group of Referees. They should be individuals who have superior knowledge and experience in the implementation and use of FORTH.

4.6.2 At least three and no more than five Referees shall be elected by a majority vote of the voting members present at the concluding general session of a formal meeting. This takes place after the election of voting members. A Referee's term is from election at the end of one formal meeting until the end of the next formal meeting. Only voting members are eligible to be elected as Referees.

D. CHARTER

4.6.3 The Referees shall adopt methods and rules as they deem appropriate to complete their work; they may be informal. However, any matter committed to the Referees for resolution must achieve near unanimous agreement (not more than one no vote). Lacking that, the matter shall be omitted from further action pending further consideration at the next formal meeting.

5. EXPERIMENTAL PROPOSALS

5.1 General

5.1.1 Since FORTH is an extensible language and subject to evolution, the Standard may contain a section describing experimental proposals. FORTH users are encouraged to study, implement, and try these proposals to aid in the analysis of and the decision for or against future adoption into the Standard. After the results of experimentation are known, each proposal will be considered, at a future formal meeting, for inclusion into the Standard.

5.1.2 An experimental proposal may be individual FORTH words, sets of related words, or specifications for part of the Standard. Experimental proposals may be derived from ordinary proposals or other contributions.

5.2 Required Information

Each experimental proposal must contain the following minimum information:

5.2.1 A description of the proposal including an overview of its functions and its interactions with existing FORTH words.

5.2.2 A glossary entry of each word in the form and notation of the Standard.

5.2.3 A statement by the author(s) indicating why the proposal merits inclusion into the Standard. Both advantages and disadvantages should be discussed.

5.3 Suggested Information

It is suggested that each experimental proposal also include:

5.3.1 A source definition for each word in the proposal. High level definitions using Standard words are preferred, but new primitive words may be defined in an assembly language of one commonly-known processor. Sufficient documentation should be provided so that implementation on other processors is direct.

5.3.2 An example showing usage of the new words.

D. CHARTER

6. VOTING

6.1 General

Only voting members have the right to vote on proposals affecting the Standard, a draft Standard, or this Charter.

6.2 Advisory Votes

At the discretion of the Chairperson, advisory votes may be requested at a formal meeting. At the discretion of the Chairperson, all attendees may participate in an advisory vote.

6.3 Method

Any vote at a formal meeting may be by show of hands or, at the discretion of the Chairperson, by an informal secret paper ballot or a roll call.

6.4 Number

A vote to adopt a proposal into the draft Standard or to change the Standard, except for the Experimental Proposals section of the Standard, requires a two-thirds affirmative vote of the voting members present at a general session of a formal meeting, provided that the number of votes cast are at least two thirds of that morning's quorum count. To adopt an experimental proposal into the Experimental Proposals section of the draft Standard or to change this Charter, an affirmative vote of a simple majority is required. Accepting any other procedural matter at a formal meeting requires only a simple majority affirmative vote.

6.5 Proxies

All votes must be cast by the particular voting member eligible to vote. No proxy voting is allowed.

E. PROPOSAL/COMMENT FORM**APPENDIX E. PROPOSAL/COMMENT FORM**

The following pages are the proposal and/or comment submittal form. The form includes instructions which should be explanatory. This form should be used for all proposals as well as comments. Copies of submitted proposals and comments will be made available to FORTH Standards Team members and to team sponsors.

Proposal and Comment Submittal Form Instructions

Please use the supplied forms for your entire proposal. The continuation form is only to be used if absolutely necessary; try to get your proposal to fit on the first sheet. If it helps, use a reducing copy machine to get more material onto the first sheet. If you must use multiple sheets, put the main idea onto the first sheet and less important material onto continuation sheets. Remember that material on continuation sheets may be overlooked.

The proposal forms have been produced on a computer system so that you may produce your proposals using your own computer system. If you print your proposal and form on your computer system, all of the information shown on the form(s) MUST be printed and in the same location.

The following are the instructions for each of the areas of the form:

1. Please think of the most appropriate keyword or keywords describing your proposal.
2. Select the best of the following categories of proposals:
 - # Nucleus Layer other than #1 (i.e., + and AND)
 - 1 Memory Operations (i.e., @ and CMOVE)
 - 2 Dictionary (i.e., ' and FORGET)
 - 3 String Operations (i.e., WORD and COUNT)
 - 4 Interpreter layer other than #2 or #3 (i.e., ABORT and .)
 - 5 Compiler Layer (i.e., : and DO)
 - 6 Device Layer (i.e., BLOCK and TYPE)
 - 7 Experimental (i.e., 32-bit stack entries)
 - 8 Other Technical (i.e., mono-addressing)
 - 9 Charter
3. Mark whether this is a PROPOSAL or a COMMENT.
4. Indicate which FORTH word or words are relevant.
5. Indicate which section or sections of the Standard are relevant.
6. The abstract must be kept short. The title, keywords, category, and abstract may be used in a database for organization and display on a terminal during a Standards Team meeting.
7. Detail your proposal and provide supporting discussion.
8. Indicate the name of the submitter or the names of the submitters.
9. Finally, date the submittal and number each page.

FST Proposal and Comment Submittal Form

```

-----
FST USE      Title:                               Proposal Number:
ONLY -->    Related Proposals:                   Disposition:
-----
Keyword(s):                                     Category:
                                           ( ) Proposal or ( ) Comment
FORTH Word(s):                               Section #(s):
-----
Abstract:

```

```

-----
Proposal and Discussion:

```

```

-----
Submitted by:                               Date:      Page    of
-----
FORTH Standards Team; PO Box 4545; Mountain View, CA 94040      820801

```

FST Proposal and Comment Submittal Continuation Form

FST USE ONLY -->

Proposal Number:

Submitted by:

Date:

Page of

FORTH Standards Team; PO Box 4545; Mountain View, CA 94040

820801

FST Proposal and Comment Submittal Form			SAMPLE
-----			SAMPLE
FST USE	Title:	Proposal Number:	SAMPLE
ONLY -->	Related Proposals:	Disposition:	SAMPLE
=====			SAMPLE
Keyword(s):	Logical True Boolean Flag	Category: 8	SAMPLE
		(*) Proposal or () Comment	SAMPLE
FORTH Word(s):	All returning a flag	Section #(s): 5	SAMPLE
-----			SAMPLE
Abstract:			SAMPLE
			SAMPLE
Standard words should return the value -1 to mean true.			SAMPLE
			SAMPLE
-----			SAMPLE
Proposal and Discussion:			SAMPLE
			SAMPLE
Proposal: Change the third sentence in the definition "true" in the section "Definitions of Terms" to read:			SAMPLE
			SAMPLE
All Standard words return a 16-bit value with all bits set to one when returning a 'true' flag.			SAMPLE
			SAMPLE
Discussion: The value of 1 representing 'true' is only applicable when it exists as a single bit. When more than one bit is used, it is more useful to set all the bits to "1" for the true condition and all the bits to "0" for the false condition. The logical operators such as AND, OR and XOR apply to each bit in a 16-bit field. The truth of logical operations follows more readily by having standard words return all bits set to 'true'. For example, <u>flag 6 *</u> in a FORTH-79 system is equivalent to <u>flag 6 AND</u> when all bits of a flag contain the same truth state; additionally <u>true 6 AND</u> in a FORTH-79 system produces a <u>false</u> . With the FORTH-79 system one could obtain the desired result by the slower operation of multiplying by the flag or by using NEGATE followed by AND which requires more memory.			SAMPLE
			SAMPLE
It appears that in almost all cases where it makes a difference, all 1's is the preferred value. Additional support for this view can be seen by examining computer hardware. We see that in most cases it is easier to obtain the all 1's state for true. In the 8086, for example, comparison operators are up to 5% faster, including the effect of NEXT, when using all 1's as a 'true' result. The only case of which we are aware that is at all slower, and then only slightly, is when using a flag result with +! to increment a variable by 1. This is, of course, balanced by using +! to decrement a variable.			SAMPLE
			SAMPLE
The negative impact of this proposal is that programs which use a flag produced by a Standard word, in other than a zero or non-zero way (i.e., with AND or *) will have to be changed. Such changes are typically trivial and tend to be both faster and use less memory.			SAMPLE
-----			SAMPLE
Submitted by: Robert Berkey, Robert L. Smith,		Date: 1 August 1982	SAMPLE
and Glenn S. Tenney		Page 1 of 1	SAMPLE
=====			SAMPLE
FORTH Standards Team; PO Box 4545; Mountain View, CA 94040		820801	SAMPLE

FORTH-79 HANDY REFERENCE

Stack inputs and outputs are shown; top of stack on right. See operand key at bottom.

STACK MANIPULATION

DUP	(n - n n)	Duplicate top of stack.
DROP	(n -)	Discard top of stack.
SWAP	(n1 n2 -> n2 n1)	Exchange top two stack items.
OVER	(n1 n2 -> n1 n2 n1)	Make copy of second item on top.
ROT	(n1 n2 n3 -> n2 n3 n1)	Rotate third item to top. "rote"
PICK	(n1 -> n2)	Copy n1-th item to top. (Thus 1 PICK = DUP, 2 PICK = OVER)
ROLL	(n -)	Rotate n-th item to top. (Thus 2 ROLL = SWAP, 3 ROLL = ROT)
?DUP	(n - n (n))	Duplicate only if non-zero. "query-dup"
>R	(n -)	Move top item to "return stack" for temporary storage (use caution). "to-r"
R>	(- n)	Retrieve item from return stack. "r-from"
R@	(- n)	Copy top of return stack onto stack. "r-fetch"
DEPTH	(- n)	Count number of items on stack.

COMPARISON

<	(n1 n2 -> flag)	True if n1 less than n2. "less-than"
=	(n1 n2 -> flag)	True if top two numbers are equal. "equals"
>	(n1 n2 -> flag)	True if n1 greater than n2. "greater-than"
0<	(n -> flag)	True if top number negative. "zero-less"
0=	(n -> flag)	True if top number zero. (Equivalent to NOT) "zero-equals"
0>	(n -> flag)	True if top number greater than zero. "zero-greater"
D<	(d1 d2 -> flag)	True if d1 less than d2. "d-less-than"
U<	(un1 un2 -> flag)	Compare top two items as unsigned integers. "u-less-than"
NOT	(flag -> -flag)	Reverse truth value. (Equivalent to 0=)

ARITHMETIC AND LOGICAL

+	(n1 n2 -> sum)	Add. "plus"
D+	(d1 d2 -> sum)	Add double-precision numbers. "d-plus"
-	(n1 n2 -> diff)	Subtract (n1-n2). "minus"
1+	(n -> n+1)	Add 1 to top number. "one-plus"
1-	(n -> n-)	Subtract 1 from top number. "one-minus"
2+	(n -> n+2)	Add 2 to top number. "two-plus"
2-	(n -> n-2)	Subtract 2 from top number. "two-minus"
*	(n1 n2 -> prod)	Multiply. "times"
/	(n1 n2 -> quot)	Divide (n1/n2). (Quotient rounded toward zero) "divide"
MOD	(n1 n2 -> rem)	Modulo (i.e., remainder from division n1/n2). Remainder has same sign as n1. "mod"
/MOD	(n1 n2 -> rem quot)	Divide, giving remainder and quotient. "ddivide-mod"
*/MOD	(n1 n2 n3 -> rem quot)	Multiply, then divide (n1*n2/n3), with double-precision intermediate. "times-divide-mod"
*/	(n1 n2 n3 -> quot)	Like */MOD, but give quotient only, rounded toward zero. "times-divide"
U*	(un1 un2 -> ud)	Multiply unsigned numbers, leaving unsigned double-precision result. "u-times"
U/MOD	(ud un -> urem uquot)	Divide double number by single, giving remainder and quotient, all unsigned. "u-divide-mod"
MAX	(n1 n2 -> max)	Leave greater of two numbers. "max"
MIN	(n1 n2 -> min)	Leave lesser of two numbers. "min"
ABS	(n -> n)	Absolute value. "absolute"
NEGATE	(n -> -n)	Leave two's complement.
DNEGATE	(d -> -d)	Leave two's complement of double-precision number. "d-negate"
AND	(n1 n2 -> and)	Bitwise logical AND.
OR	(n1 n2 -> or)	Bitwise logical OR.
XOR	(n1 n2 -> xor)	Bitwise logical exclusive-OR. "x-or"

MEMORY

@	(addr → n)	Replace address by number at address. "fetch"
!	(n addr →)	Store n at addr. "store"
C@	(addr → byte)	Fetch least significant byte only. "c-fetch"
CI	(n addr →)	Store least significant byte only. "c-store"
?	(addr →)	Display number at address. "question-mark"
+!	(n addr →)	Add n to number at addr. "plus-store"
MOVE	(addr1 addr2 n →)	Move n numbers starting at addr1 to memory starting at addr2, if n>0.
CMOVE	(addr1 addr2 n →)	Move n bytes starting at addr1 to memory starting at addr2, if n>0. "c-move"
FILL	(addr n byte →)	Fill n bytes in memory with byte beginning at addr, if n>0.

CONTROL STRUCTURES

DO ... LOOP	do: (end+1 start →)	Set up loop, given index range.
I	(→ index)	Place current loop index on data stack.
J	(→ index)	Return index of next outer loop in same definition.
LEAVE	(→)	Terminate loop at next LOOP or +LOOP, by setting limit equal to index.
DO ... +LOOP	do: (limit start →)	Like DO ... LOOP, but adds stack value (instead of always 1) to index. Loop terminates when index is greater than or equal to limit (n>0), or when index is less than limit (n<0). "plus-loop"
IF ... (true) ... THEN	if: (flag →)	If top of stack true, execute.
IF ... (true) ... ELSE	if: (flag →)	Same, but if false, execute ELSE clause.
... (false) ... THEN		
BEGIN ... UNTIL	until: (flag →)	Loop back to BEGIN until true at UNTIL.
BEGIN ... WHILE	while: (flag →)	Loop while true at WHILE; REPEAT loops unconditionally to BEGIN. When false, continue after REPEAT.
... REPEAT		
EXIT	(→)	Terminate execution of colon definition. (May not be used within DO ... LOOP)
EXECUTE	(addr →)	Execute dictionary entry at compilation address on stack (e.g., address returned by FIND).

<i>Operand key:</i>	d, d1, ...	32-bit signed numbers	addr, addr1, ...	addresses	char	7-bit ascii character value
n, n1, ...	u	unsigned	byte	8-bit byte	flag	boolean flag

TERMINAL INPUT-OUTPUT

CR	(→)	Do a carriage return and line feed. "c-r"
EMIT	(char →)	Type ascii value from stack.
SPACE	(→)	Type one space.
SPACES	(n →)	Type n spaces, if n>0.
TYPE	(addr n →)	Type string of n characters beginning at addr, if n>0.
COUNT	(addr → addr+1 n)	Change address of string (prefixed by length byte at addr) to TYPE form.
-TRAILING	(addr n1 → addr n2)	Reduce character count of string at addr to omit trailing blanks. "dash-trailing"
KEY	(→ char)	Read key and leave ascii value on stack.
EXPECT	(addr n →)	Read n characters (or until carriage return) from terminal to address, with null(s) at end.
QUERY	(→)	Read line of up to 80 characters from terminal to input buffer.
WORD	(char → addr)	Read next word from input stream using char as delimiter, or until null. Leave addr of length byte.

NUMERIC CONVERSION

BASE	(→ addr)	System variable containing radix for numeric conversion.
DECIMAL	(→)	Set decimal number base.
.	(n →)	Print number with one trailing blank and sign if negative. "dot"
U.	(un →)	Print top of stack as unsigned number with one trailing blank. "u-dot"
CONVERT	(d1 addr1 → d2 addr2)	Convert string at addr1 + 1 to double number. Add to d1 leaving sum d2 and addr2 of first non-digit.
<#	(→)	Start numeric output string conversion. "less-sharp"
#	(ud1 → ud2)	Convert next digit of unsigned double number and add character to output string. "sharp"
#S	(ud → 0 0)	Convert all significant digits of unsigned double number to output string. "sharp-s"
HOLD	(char →)	Add ascii char to output string.
SIGN	(n →)	Add minus sign to output string if n<0.
#>	(d → addr n)	Drop d and terminate numeric output string, leaving addr and count for TYPE. "sharp-greater"

MASS STORAGE INPUT/OUTPUT

LIST	(n -)	List screen n and set SCR to contain n.
LOAD	(n -)	Interpret screen n, then resume interpretation of the current input stream.
SCR	(- addr)	System variable containing screen number most recently listed.
BLOCK	(n - addr)	Leave memory address of block, reading from mass storage if necessary.
UPDATE	(-)	Mark last block referenced as modified.
BUFFER	(n - addr)	Leave addr of a free buffer, assigned to block n; write previous contents to mass storage if UPDATED.
SAVE-BUFFERS	(-)	Write all UPDATED blocks to mass storage.
EMPTY-BUFFERS	(-)	Mark all block buffers as empty, without writing UPDATED blocks to mass storage.

DEFINING WORDS

: xxx	(-)	Begin colon definition of xxx . "colon"
:	(-)	End colon definition. "semi-colon"
VARIABLE xxx	(-)	Create a two-byte variable named xxx ; returns address when executed.
	xxx: (- addr)	
CONSTANT xxx	(n -)	Create a constant named xxx with value n; returns value when executed.
	xxx: (- n)	
VOCABULARY xxx	(-)	Create a vocabulary named xxx ; becomes CONTEXT vocabulary when executed.
CREATE ... DOES>	does: (- addr)	Used to create a new defining word, with execution-time routine in high-level FORTH. "does"

VOCABULARIES

CONTEXT	(- addr)	System variable pointing to vocabulary where word names are searched for.
CURRENT	(- addr)	System variable pointing to vocabulary where new definitions are put.
FORTH	(-)	Main vocabulary, contained in all other vocabularies. Execution of FORTH sets context vocabulary.
DEFINITIONS	(-)	Sets CURRENT vocabulary to CONTEXT.
' xxx	(- addr)	Find address of xxx in dictionary; if used in definition, compile address. "tick"
FIND	(- addr)	Leave compilation address of next word in input stream. If not found in CONTEXT or FORTH, leave 0.
FORGET xxx	(-)	Forget all definitions back to and including xxx , which must be in CURRENT or FORTH.

COMPILER

,	(n -)	Compile a number into the dictionary. "comma"
ALLOT	(n -)	Add two bytes to the parameter field of the most recently-defined word.
."	(-)	Print message (terminated by "). If used in definition, print when executed. "dot-quote"
IMMEDIATE	(-)	Mark last-defined word to be executed when encountered in a definition, rather than compiled.
LITERAL	(n -)	If compiling, save n in dictionary, to be returned to stack when definition is executed.
STATE	(- addr)	System variable whose value is non-zero when compilation is occurring.
[(-)	Stop compiling input text and begin executing. "left-bracket"
]	(-)	Stop executing input text and begin compiling. "right-bracket"
COMPILE	(-)	Compile the address of the next non-IMMEDIATE word into the dictionary.
[COMPILE]	(-)	Compile the following word, even if IMMEDIATE. "bracket-compile"

MISCELLANEOUS

((-)	Begin comment, terminated by) on same line or screen; space after (. "paren", "close-paren"
HERE	(- addr)	Leave address of next available dictionary location.
PAD	(- addr)	Leave address of a scratch area of at least 64 bytes.
>IN	(- addr)	System variable containing character offset into input buffer; used, e.g., by WORD. "to-in"
BLK	(- addr)	System variable containing block number currently being interpreted, or 0 if from terminal. "b-l-k"
ABORT	(-)	Clear data and return stacks, set execution mode, return control to terminal.
QUIT	(-)	Like ABORT , except does not clear data stack or print any message.
79-STANDARD	(-)	Verify that system conforms to FORTH-79 Standard.

Index

!	3	(?DO)	29	/LOOP	62
!CSP	3	(?LEAVE)	30	/MOD	63
!L	4	(ABORT)	31	0	64
"	5	(D.)	31	0<	65
#	6	(DO)	32	0=	66
#>	7	(FIND)	33	0>	66
#BUFF	7	(LINE)	35	0BRANCH	67
#LINE	8	(LOOP)	36	1	68
#OUT	9	(NUMBER)	37	1+	69
#S	9	(S	38	1-	69
#THREADS	10	(U.)	38	2	70
#TIB	11	*	39	2!	71
'	11	*/	40	2*	71
'-FIND	12	*/MOD	41	2+	72
'?TERMINAL	13	+	42	2-	73
'ABORT	14	+!	42	2/	73
'BLOCK	14	+ -	43	2@	74
'CR	15	+BUF	44	2CONSTANT	75
'EMIT	16	+LOOP	45	2DROP	76
'EXPECT	16	+ORIGIN	46	2DUP	77
'INTERPRET	17	,	47	2OVER	77
'KEY	18	, "	48	2ROT	78
'LOAD	18	-	48	2SWAP	79
'NUMBER	19	-->	49	2VARIABLE	80
'PAGE	20	-DUP	50	79-STANDARD	81
'R/W	20	-FIND	51	:	81
'S	21	-ROT	52	;	83
'STREAM	21	-TEXT	52	;CODE	84
'T&SCALC	22	-TRAILING	53	;S	85
'TIB	23	.	55	<	86
'TITLE	23	."	55	<#	87
'VOCABULARY	24	.(57	<+LOOP>	88
'WORD	25	.ID	57	<-FIND>	88
(25	.INDEX	58	<.">	89
(.)	27	.LINE	59	</LOOP>	90
(. ")	27	.R	59	<;CODE>	91
(+LOOP)	28	.S	60	<<CMOVE>	92
(;CODE)	29	/	61	<>	92

<?TERMINAL>	93	>VIEW	125	BLANK	152
<ABORT">	94	?	125	BLANKS	153
<ABORT>	95	?BRANCH	126	BLK	153
<BLOCK>	95	?COMP	127	BLK/DRV	154
<BUILDS	96	?CONFIGURE	127	BLOCK	155
<CMOVE	97	?CSP	129	BLOCK-READ	156
<CMOVE>	98	?DO	129	BMOVE	157
<CR>	99	?DUP	130	BODY>	157
<DO>	100	?ERROR	131	BOOT	158
<EMIT>	100	?EXEC	131	BOUNDS	159
<EXPECT>	101	?LOADING	132	BPDRV	159
<FILL>	102	?PAIRS	133	BRANCH	160
<FIND>	103	?STACK	133	BS	161
<INTERPRET>	104	?STREAM	135	BUFFER	162
<KEY>	105	?TERMINAL	135	BYE	163
<LINE>	106	@	136	C!	164
<LOAD>	106	@L	137	C!L	165
<LOOP>	107	ABORT	138	C,	165
<MARK	108	ABORT"	138	C/L	166
<NUMBER>	109	ABS	139	C@	167
<PAGE>	110	AGAIN	140	C@L	168
<R/W>	111	ALLOT	141	CAPS	168
<RESOLVE	112	ALSO	142	CASE	169
<T&SCALC>	113	AND	142	CFA	170
<VOCABULARY79>	114	APUSH	143	CHANGE	171
<VOCABULARYFIG>	115	ASSEMBLER	144	CLEAR	172
<WORD>	116	ASCII	144	CMOVE	173
=	117	B/BUF	145	CMOVE>	174
>	117	B/SCR	146	CODE	175
>BINARY	118	BACK	146	COLD	176
>BODY	119	BASE	146	COMPARE	177
>IN	120	BDOS	147	COMPILE	178
>LINK	120	BEEP	148	CONFIGURE	178
>MARK	121	BEGIN	148	CONSTANT	180
>NAME	121	BELL	150	CONTEXT	181
>R	122	BETWEEN	150	CONVERT	182
>RESOLVE	123	BINARY	151	COPY	183
>TYPE	124	BL	152	COUNT	183

INDEX**APPENDIX I**

CR	184	DPL	215	FORTH	247
CREATE	185	DPUSH	216	FORTH-83	248
CSP	187	DR-DEN	217	FREEZE	249
CURRENT	188	DR0	217	GO	249
D!	188	DRIVE	218	H	250
D+	189	DROP	219	HERE	251
D+-	190	DSWAP	219	HEX	251
D-	190	DU<	220	HIDE	252
D.	191	DUMP	221	HLD	253
D.R	192	DUP	222	HOLD	253
D0=	193	DVARIABLE	223	HPUSH	254
D<	194	EDITOR	224	I	255
D=	194	ELSE	224	I'	255
D>	195	EMIT	225	ID.	256
D@	196	EMPTY	226	IF	257
DABS	196	EMPTY-BUFFERS	227	IMMEDIATE	258
DCONSTANT	197	ENCLOSE	228	IN	259
DDROP	198	END	229	INDEX	260
DDUP	199	END-CODE	230	INIT-FORTH	261
DECIMAL	199	ENDCASE	230	INIT-USER	261
DEFER	200	ENDIF	232	INTCALL	262
DEFINITIONS	201	ENDOF	232	INTERPRET	263
DEN	202	ENTRY	233	IS	264
DENSITY	202	EPRINT	234	J	265
DEPTH	203	ERASE	235	KEY	266
DIGIT	204	ERROR	235	KEY?	267
DISK-ERROR	205	EXECUTE	236	L!	268
DLIST	206	EXIT	237	L@	268
DLITERAL	206	EXPECT	238	LATEST	269
DMAX	207	FALSE	239	LEAVE	269
DMIN	208	FENCE	240	LFA	271
DMINUS	208	FILL	241	LIMIT	271
DNEGATE	209	FIND	241	LINK	272
DO	210	FIRST	243	LINK>	273
DOES>	211	FLD	243	LIST	273
DONE?	213	FLIP	244	LIT	274
DOVER	214	FLUSH	244	LITERAL	275
DP	214	FORGET	245	LOAD	276

INDEX**APPENDIX I**

LOOP	277	PC!	306	SAVE-FORTH	334
M*	278	PC@	307	SCAN	335
M*/	279	PAD	307	SCR	336
M+	280	PAGE	308	SEAL	337
M/	280	PAUSE	309	SEC	337
M/MOD	281	PC!	310	SEC-READ	338
MAX	282	PC@	310	SEC-WRITE	339
MAX-DRV	283	PERFORM	311	SEC/BLK	340
MESSAGE	284	PFA	312	SEC/DR	341
MIN	285	PICK	312	SEC/TR	342
MINUS	285	POP	313	SEE	342
MOD	286	PP	314	SET-DRIVE	343
MON	287	PREV	314	SET-DRX	344
MOVE	287	PUSH	315	SET-IO	345
MU/MOD	288	PUT	315	SIGN	346
MYSELF	289	PW!	316	SKIP	346
NAME>	289	PW@	317	SMUDGE	347
NEGATE	290	QUERY	317	SP!	348
NEXT	291	QUIT	318	SP0	349
NEXT1	292	R	320	SP@	350
NFA	292	R#	320	SPACE	350
NIP	293	R/W	321	SPACES	351
NOOP	293	R0	322	SPAN	352
NOT	294	R>	322	SPBLK	352
NUMBER	295	R@	323	SPDRV	353
NUMBER?	296	RECURSE	324	SPLIT	354
OCTAL	297	RECURSIVE	325	SPT	354
OF	297	REPEAT	325	STATE	355
OFF	298	REVEAL	326	SWAP	356
OFFSET	299	ROLL	327	SYSCALL	356
ON	300	ROT	328	T&SCALC	357
ONLY	300	RP!	329	TASK	358
OR	301	RP@	330	TEXT	359
ORDER	302	RPP	330	THEN	360
OUT	303	S->D	331	THRU	360
OVER	303	S0	332	TIB	361
PI	304	S>D	332	TITLE	362
P@	305	SAVE-BUFFERS	333	TO	363

TOGGLE	363	UMAX	376	WHILE	390
TOS	364	UMIN	376	WIDTH	392
TRACK	365	UNTIL	377	WITHIN	392
TRAVERSE	365	UP	378	WORD	393
TRIAD	366	UPDATE	379	WORDS	395
TRUE	367	USE	380	X	396
TUCK	368	USER	381	XOR	397
TYPE	368	VALUE	382	[398
U*	369	VARIABLE	382	[']	399
U.	370	VIEW	384	[COMPILE]	400
U.R	371	VLIST	385	\	401
U/	372	VOC-LINK	386	\S	402
U/MOD	372	VOCABULARY	386]	402
U<	373	WARM	388		
UM*	374	WARNING	389		
UM/MOD	375	WHERE	389		